Netsite™ Commerce Server Administrator's Guide

Document Number 007-2666-001

Netsite™ Commerce Server Administrator's Guide
Document Number 007-2666-001

Contents

List of Examples

List of Figures

About This Guide

Welcome to the world of WebFORCE™. You're about to embark on a fascinating journey. You've chosen the best network and Internet communications server available, and you'll discover it's quite easy to configure, install, and manage.

This reference guide tells you everything you need to know about getting started with the Netsite™ Commerce Server and then how to reconfigure and maintain it. The guide is organized in much the same way as the online forms you use to install and manage the server. Chapter 1 tells you how to do the setup and install. Chapter 2 shows you how to gain access to the Netsite Server Manager. Chapter 3 shows you how to use the Netsite Server Manager to perform additional server configuration. Chapter 4 discusses server security issues. Chapter 5 tells you how to do server maintenance. The appendices describe what's new and different about Netsite, provide valuable tutorials and show you how to work directly with the server's configuration files should you choose to do so.

Contact Information

Silicon Graphics, Inc., provides a comprehensive product support maintenance program for its products.

If you are in the United States or Canada and would like support for your Silicon Graphics-supported products, contact the Technical Assistance Center at 1-800-800-4SGI.

If you are outside these areas, contact the Silicon Graphics subsidiary or authorized distributor in your country.

Some Conventions Used in this Guide

Every computer reference guide has conventions. There's no avoiding them. They'll help you find your way through the reference guide.

Note: This convention is a note. It provides special information that you might not otherwise see or pay any attention to.

This book describes, for the most part, how you fill out online document forms. So when this guide mentions "form" or "page", it means the document form or page that you'd see online if you were sitting in front of your workstation looking at the monitor while working with the Netsite server software (not the pages or sample forms in the guide—unless it is specifically stated otherwise).

When forms input is talked about in the guide, you'll see something like this:

Characters that you type on your keyboard appear in bold type and look **`like this`**.

1. Numbered steps have numbers like the one at the beginning of this sentence.

Uniform Resource Locators (URLs) are set off from the body of certain paragraphs and look like this:

`http://www.mcom.com`

Items that appear between brackets like `[yourdomain]` should be replaced with a value or expression specific to what you're doing or specific to your site.

Variables that are used in syntax expressions or descriptions of syntax expressions appear in a typestyle that looks *like this*. This convention is used primarily in Appendix C.

With this information about some of the conventions used in this guide now embedded in your mind, you're ready to start installing your Netsite Communications Server. Read on.

Chapter 1

Quickstart Installation

It's easy to get your Netsite Commerce Server up and running. There are only five steps to the QuickStart process:

1. Answer the questions on the configuration worksheet found in Figure 1-3. This chapter has the answers to those questions.
2. Load the software onto your machine as described in the release notes.
3. Fill out the online Netsite configuration forms (using your answers to the questions on the worksheet).
4. Register your server with the Netscape Communications Directory of Services (optional).
5. Click a link that automatically installs the server to your specs.

To load the software onto your server host, follow the instructions for *inst* contained in your software released notes. When you have loaded that software and run the *./mc-install* command (thereby starting the *installer*), you are presented with the QuickStart forms described in this chapter.

Use the worksheet (Figure 1-3) and this guide to plan your configuration and get all the information you need in one place. Then, when you go to your workstation to perform the initial server setup, you may be surprised at how easy it goes.

Things You Can Do Before You Do Anything Else

To prepare for the server setup and configuration, take some time to look over these items. If you do these things before you do anything else, the server installation will go smoothly. And, you'll have everything ready to go when the time comes.

Make Sure DNS Is Up and Running

When you do the Netsite Communications Server installation, some items on the installation forms request either a hostname or an IP address (or multiple entries of the same) as input strings. As you prepare for installation, if you plan to use hostnames, make sure your domain name system (DNS) service is up and running properly. Otherwise, the Netsite server won't be able to resolve hostnames when it's installed and running. If you can't use DNS or don't have name service running, make sure you use full IP addresses instead of hostnames where requested on the forms.

Make a Home Page

If you've done any "netsurfing" on the Internet with Netscape Navigator™ or some other network navigation software, you've probably seen these. A home page serves as an introduction or index to the contents of your server. You might already have one. If you have a home page and want to use it with the new server, make a mental note of it and skip to the next part of the guide. If you don't have a home page, you probably want to make one. If you don't know how or don't have the time right now, the server installer will make one for you, but it's short on glamour. If you want to bring up your Netsite service in the most elegant manner possible, having a home page ready to go advances this cause.

Think of a Name for the Server and Set Up an Alias

Every "thing" needs a name. In the world of computers and networks, if something isn't named, it doesn't exist. You need to come up with a name for your server so that people know it exists and how to find it.

When the Netsite server is installed, the name you give your server, combined with your domain name and domain authority, becomes the Uniform Resource Locator (URL) when users attempt to access your home page. For example, when people use Netscape Navigator to reach us, they use the URL:

```
http://www.sgi.com
```

You can see that the server name combines our domain name, “sgi” with the top-level domain authority “com”, with a server name of “www” (separated by the appropriate dots).

Using Netscape Navigator, you can do some netsurfing on your own and take a look at the various URLs out there. They should give you an idea of what to call your own server. Keep in mind that the server name part of a URL—the last three strings between the dots—looks like this:

```
servername.[yourdomain].[dom]
```

Look at Figure 1-1 to see a visual description of URLs. For more examples of server names see “What do you want to name your server?” on page 9.

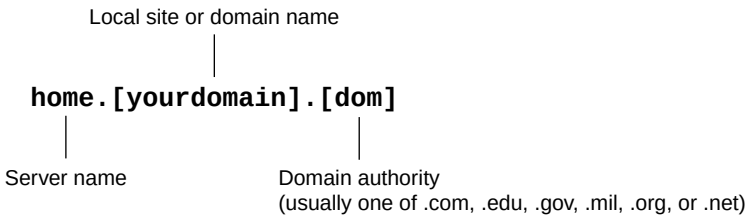


Figure 1-1 Anatomy of a URL

DNS service must be supported on the server machine and all referring machines in the network if you plan to use a hostname. Otherwise, you need to make your server name using the exact IP address of the machine where you plan to install the server. For example, the IP address for the hostname www.sgi.com is 192.82.208.8.

A domain name system service matches the IP address with the domain name using a database of machine name records and IP address records. If you have any doubt that DNS is supported on the machine where you plan to install your Netsite server, use your machine’s IP address for the server hostname. IP addresses always work.

If your server is going to run on one machine among many in a network, you or your network administrator should set up a DNS CNAME record or an *alias* like “www” which points to the actual server machine. Later, should the need arise, you can change the actual hostname or IP address of the server machine without having to change all of your URLs.

Create an IRIX User Account

You need to be logged in as root (or superuser) to install the server. However, you don't necessarily want the server to run as root all the time. You probably want the server to have restricted access to your system resources and run under a non-privileged system user account. In this case, you need to create an IRIX user account for the server. When the server launches, it runs as this user. Likewise, any child processes of the server are created with this server user as the process owner.

You can choose the user *nobody* if you wish, but this might not work on some systems. Some machines ship with a *uid* of *-2* for the user *nobody.* A *uid* less than zero generates an error during installation. Check the */etc/passwd* file to see if the *uid* for *nobody* exists and that it is greater than zero.

If you'd prefer to use a different account than *nobody*, just create and use a regular IRIX user account. (If you don't know how to create a new user on your system, refer to the *IRIX Personal Systems Administration Guide.*)

Create a Directory to Serve as Your Document Root

Using the Netsite server, you can keep all your documents in a single tree of directories. When you set up the server during the installation, you tell it where the root of this directory tree, called the "document root," is located. Although the server installer can create one during the configuration, it's a good idea to have one already. If you make a directory for your documents, make sure the system user account has *read* and *execute* privileges to access it.

Choose a Unique Port Number

Port numbers for all network accessible services are maintained in the */etc/services* file. Just as the standard Telnet port number is 23, the standard HTTP port number is 80. However, if you plan to put your server on a port other than 80, you should make sure that port isn't already being used. If you have any doubts about whether the port you plan to assign the server is unique, look at */etc/services* on the server machine to make sure you don't assign a port number that is already taken by another service.

Deactivate Any Existing HTTP Servers

If you plan to install the Netsite server using the same port number as the HTTP server you're now running, you need to deactivate the existing HTTP server. If you want to install the new server in the same directory location as your existing server, you should move the existing server to another directory. If you use the same directory as your existing HTTP server when setting up the Netsite server root, the installer can overwrite your existing HTTP server. If you use the same port number, the Netsite server can't start itself using the server installer.

What Happens During Installation?

You load the Netsite server software onto your machine with the *inst* program as described in your software release notes. You run the Netsite server installer program (*mc-install*). You fill out the online document forms and submit them. Figure 1-2 provides an overview of the forms pages you fill out. When the forms are complete, you click a link that installs the server. During the initial setup you tell the installer where to put the server. These items (three directories, two scripts and the server application) are installed in that location:

```
admin  kill-httpd  logs  mc-httpd  mc-icons  start-httpd
```

kill-httpd is the script to use if you have to start the server from the command line. *mc-httpd* is the server application. *start-httpd* is the script you use if you have to start the server from the command line.

The server writes the configuration files into the appropriate directories and starts the server daemon running. If you specify a document root directory and it does not exist, the server installer creates one. That's all there is to it.

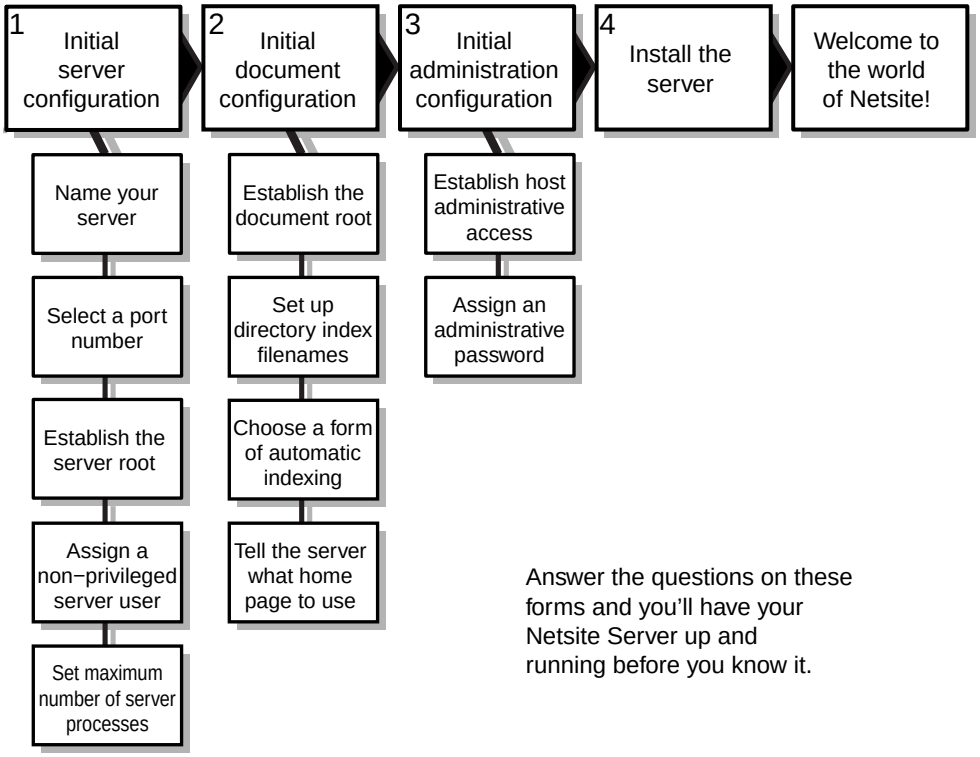


Figure 1-2 A Map of the Online Installation Forms

Server Configuration

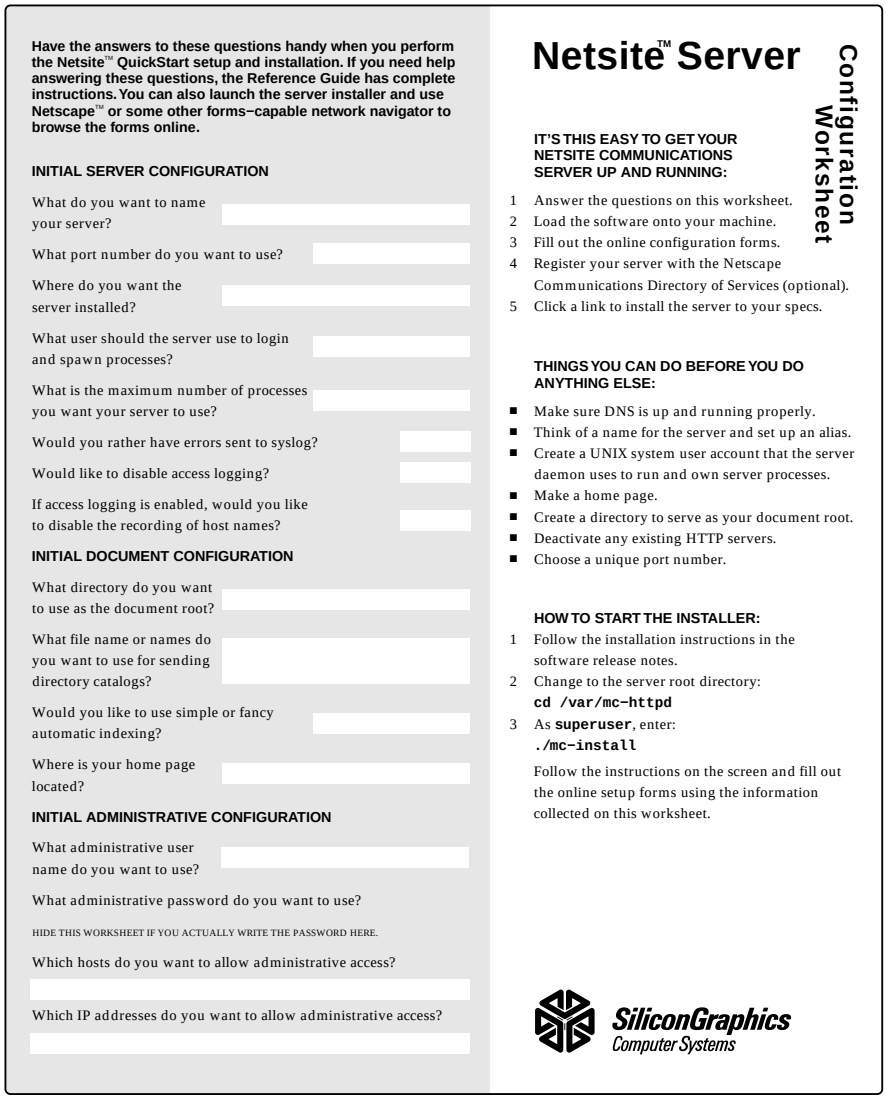
Once you have installed the software and started the server installer as described in the release notes, you are prompted to enter the name of the HTTP-compatible network navigation software you plan to use. Silicon Graphics recommends Netscape Navigator. If you do not have Netscape Navigator, please use NCSA Mosaic for the X Window System™ version 2.4, as it is known to have no problems managing this server product.

After typing the name (or accepting the displayed default) and pressing **<Enter>**, you can begin the setup and installation. Read through the

following sections of this reference guide to make sure your configuration worksheet is prepared properly.

Fill Out the Configuration Worksheet

If you take the time to answer the questions in the configuration worksheet (shown in Figure 1-3), you'll save time when you go to do the installation. All the necessary information will be handy and in one place. You can then sit at your workstation and fill out the online forms using the information already collected. This part of the guide helps you answer the worksheet questions based on the questions used by the Netsite server installation forms.



Have the answers to these questions handy when you perform the Netsite™ QuickStart setup and installation. If you need help answering these questions, the Reference Guide has complete instructions. You can also launch the server installer and use Netscape™ or some other forms−capable network navigator to browse the forms online.

INITIAL SERVER CONFIGURATION

What do you want to name your server?

What port number do you want to use?

Where do you want the server installed?

What user should the server use to login and spawn processes?

What is the maximum number of processes you want your server to use?

Would you rather have errors sent to syslog?

Would like to disable access logging?

If access logging is enabled, would you like to disable the recording of host names?

INITIAL DOCUMENT CONFIGURATION

What directory do you want to use as the document root?

What file name or names do you want to use for sending directory catalogs?

Would you like to use simple or fancy automatic indexing?

Where is your home page located?

INITIAL ADMINISTRATIVE CONFIGURATION

What administrative user name do you want to use?

What administrative password do you want to use?

HIDE THIS WORKSHEET IF YOU ACTUALLY WRITE THE PASSWORD HERE.

Which hosts do you want to allow administrative access?

Which IP addresses do you want to allow administrative access?

Netsite™ Server Configuration Worksheet

IT'S THIS EASY TO GET YOUR NETSITE COMMUNICATIONS SERVER UP AND RUNNING:

1 Answer the questions on this worksheet.
2 Load the software onto your machine.
3 Fill out the online configuration forms.
4 Register your server with the Netscape Communications Directory of Services (optional).
5 Click a link to install the server to your specs.

THINGS YOU CAN DO BEFORE YOU DO ANYTHING ELSE:

- Make sure DNS is up and running properly.
- Think of a name for the server and set up an alias.
- Create a UNIX system user account that the server daemon uses to run and own server processes.
- Make a home page.
- Create a directory to serve as your document root.
- Deactivate any existing HTTP servers.
- Choose a unique port number.

HOW TO START THE INSTALLER:

1 Follow the installation instructions in the software release notes.
2 Change to the server root directory:
`cd /var/mc−httpd`
3 As **`superuser`**, enter:
`./mc−install`

Follow the instructions on the screen and fill out the online setup forms using the information collected on this worksheet.

SiliconGraphics Computer Systems

Figure 1-3 Sample Configuration Worksheet

Initial Server Configuration

The first form you see sets up and controls the initial Netsite Communications Server configuration. Answering the questions on this page, you'll:

- Name your server
- Select a port number
- Establish the server root
- Assign a non-privileged server user
- Set the maximum number of processes the server will spawn
- Set server log options

All the questions on the online page have default entries or values. You can use those or put your preferred answers on the configuration worksheet using the guidelines below.

What do you want to name your server?

To answer this question properly you need to understand that when the Netsite server is installed, the name you give your server, combined with your domain name and domain authority, becomes the Uniform Resource Locator (URL) when users attempt to access your home page. For example, when people use Netscape Navigator to reach Silicon Graphics, they use the URL:

```
http://www.sgi.com
```

As expressed previously, you can see that the server name combines our domain name, "sgi" with the top-level domain authority "com", with a server name of "www" (separated by the appropriate dots). This is important if your server is going to be accessed by outside clients. Make sure you use a name like

```
www.[yourdomain].[dom]
```

and not just "www" without the domain name (see Figure 1-1). As stated before, DNS service must be supported on the server machine and all referring machines in the network if you plan to use a hostname. Otherwise,

you need to build the URL to your server using the IP address of the machine.

Keep in mind, if your server is going to run on one machine among many in a network, you or your system administrator should have set up a DNS CNAME record or an *alias* like "www" which points to the actual server machine. Later, should the need arise, you can change the actual hostname or IP address of the server machine without changing all of your URLs.

Here are some examples of server names:

- `www.university.edu`
- `www.company.com`
- `www.agency.gov`

What port number do you want to use?

To answer this question, you need to know a bit about port numbers and how processes bind to a port number when they are launched. As mentioned earlier, the standard port number for HTTP servers is 80. If you plan to run a server that allows access from the world at large, it's a good idea to stick with this common port number assignment.

When you complete the Netsite Communications Server install, the server launches and binds to the port number you assign at this point in the setup. Since you are logged in as either root or have superuser status while doing the install, the server can bind to any port across the complete range of port numbers available (1 to 65535) that are not being used by another service on the machine. Under normal operation, a server that binds to ports 1 through 1024 must be launched by a process with root or superuser privileges. A server that binds to a port in the range 1025 to 65535 can be launched by a non-privileged process.

If the port number you assign to the server is in the privileged range (1–1024) and you should ever need to take the server completely offline, you'd need to again log in as root or attain superuser status to restart it from the command line. This situation is rare.

On IRIX machines, the */etc/services* file maintains the list of port numbers for all network accessible services. If you don't want to use port number 80,

examine the */etc/services* file to make sure you pick an unused port. If you end up choosing a port number other than 80, you'll have to include that port number in URLs used to access your server. For example, if you name your server:

```
www.[yourdomain].[dom]
```

and give it port number 8080, the initial URL pointing to your server would have to be:

```
http://www.[yourdomain].[com]:8080/
```

Where do you want the server installed?

Note: If you have an existing HTTP server still running, don't use a path that points to the same directory as your existing HTTP server.

The Netsite Communications Server and its support files get installed in a central directory you name here. This directory contains the server program, log files, configuration files, and administrative forms. The installer creates this directory if it does not already exist.

Here are some examples of common directory locations for the server software:

- */var/mc-httpd*
- */usr/mc-httpd*
- */usr/people/mc-httpd*

What user should the server use to log in and spawn processes?

As suggested earlier, the name you write on your worksheet here and enter into the online forms during setup should already exist as a normal IRIX system user. When the server daemon starts, it runs as this system user. Likewise, any child processes spawned by the server are created with this user as the process owner. Normally, you want the server to have restricted access to your system resources and run as a non-privileged user. Even though you're logged in as root when you set up and install the server, the system user account the server uses need not be privileged in order to work properly.

The default user for this field in the online form is nobody. As mentioned previously, you can accept the default user nobody if you wish, but this might not work on all systems. Some machines ship with a uid of -2 for the user nobody. The uid of a user chosen here must be greater than zero. A UID less than zero generates an error during installation. You can check the */etc/passwd* file to see if the UID for nobody exists and that it is greater than zero.

What is the maximum number of processes you want your server to use?

Whenever people access your server using an HTTP-based client, the Netsite server uses background processes to service the requests. The processes are spawned when the server starts. They remain idle until needed. This number can be changed at any time using the Netsite Server Manager forms. You can set the number of processes to any value from 1 up to a maximum of 1024.

Base your choice here on achieving a balance between system load and server requests. Choosing too many processes could cause needless memory swapping as a large number of processes could grow to consume available RAM. On the other hand, choosing too few can cause delays when users attempt to access your server.

If your server operates with fairly low demand (under 10,000 accesses per day), or if your server is being used for other jobs besides httpd service, running 8-16 processes should be enough. If your server operates with higher demand, you may want to make sure you have enough RAM and set the number as high as you can. Thirty-two should be enough for most high access sites, with 48 or 64 being even better if you have the RAM to spare.

To illustrate what happens if you choose a number of processes which is too low, suppose that you've chosen some number of processes for your machine. Suppose also that some number of people are currently connected to your server and downloading large files. When another client accesses your server, that client is placed in a waiting queue and its request is delayed until another process becomes available. If you set the maximum number of processes way too low, then clients will begin to receive timeout errors after a certain number of them are in the queue.

Choosing the proper number of processes is mostly a matter of fine-tuning. The easiest way to tell how your server is doing is to try accessing it during peak hours to see what kind of response time you get. You can also use system-specific tools such as *top* to see how much memory your server is using.

Would you rather have errors sent to syslog?

Would you like to disable access logging?

If access logging is enabled, would you like to disable the recording of hostnames?

These three questions are check box items. If the items are left unchecked, the default answer to each question is "no." To answer "yes," click in one of the boxes. Otherwise, leave the box empty. On your worksheet, you can check the item or write "yes" for each option you want to use.

The answers to these three questions can affect system performance. Under normal operation, the Netsite server maintains two log files in your server root directory (which you named earlier on this form). One file logs server errors; the other keeps a log of who accesses your server. You can have errors reported through your system's syslog facility if you wish. Otherwise, errors are sent to a log in the server root directory, which is easily viewable using the Netsite Server Manager.

If you decide to keep access logging enabled, the server maintains a log of accesses by hostnames. You may need extra disk space to accommodate the logging of hostnames. However, if DNS or your name service application isn't up and running properly, you'll see an error message in the log file for each hostname the server can't resolve. If you don't want hostnames recorded, you need to check the box next to the question that disables the recording of hostnames. The access log will continue to log accesses by IP address.

If you record hostnames, server processes can get bogged down doing DNS lookups. You can imagine how this would work if you are or plan to be a high-demand site. This can slow performance on your entire system. The alternative is to do access logging but leave the recording of hostnames

turned off. If you ever do need to identify a particular host, get the IP address from the log and use an IP resolver utility to get the hostname.

Submitting the Form

Once this form is all filled out and you're happy with your entries, click the "Submit this form" button. If everything you entered is valid, the next page you see says, "Success!" and you're presented with the opportunity to quit the install or continue on to the next form. If the entries weren't valid, you'll be asked to do the form over again.

Initial Document Configuration

The second form you see using the server installer sets up and controls the initial Netsite server document configuration. Answering the questions on this page, you'll:

- Establish the document root
- Set up directory index filenames
- Choose a form of automatic indexing
- Tell the server what home page to use

All the questions on the online page have default entries or values. You can use those or put your preferred answers on the configuration worksheet using the guidelines below.

What directory do you want to use as the document root?

By creating a root directory for all of your documents, you can keep all your documents in one location and let the server handle the URLs. This way, any incoming request for a document automatically gets redirected to the document root directory you name here. Full file system pathnames are not used and are not displayed on any HTTP-compatible client. This keeps your file system safe from outsiders who won't be able to get any information about the rest of your system.

Using a central document root directory also lets you move your documents to a larger disk as your Netsite service grows and expands, without having to change your URLs.

Here are some examples of document root directories:

- */usr/netsite-docs*
- */usr/html-docs*
- */usr/content*
- */netsite-pages*

The installer can create this directory with a default name of */usr/netsite-docs* if one does not already exist. If you make the directory, make sure the server system user has *read* and *execute* privileges to access it.

What filename or names do you want to use for sending directory catalogs?

When you reference a directory on your server, it's helpful to have an index file in it that tells people what's in the directory. When people follow the URL that points to a directory, the server finds this file and uses it to display a catalog of what's inside. By entering a name here, you can standardize directory index filenames. A popular choice is *index.html* (the default supplied in the online form). If you want to use more than one name, separate the names with commas. The server sends back the first one it finds. Spaces between words in the filename are ignored.

Would you like to use simple or fancy automatic indexing?

The Netsite server creates an index of directory contents automatically every time a directory is accessed that doesn't have an index file with one of the names you entered above.

These automatic indexes come in two flavors, simple and fancy. A simple index displays a list of the directory contents by name only. A fancy index also displays icons, file sizes, and last modification dates.

There are buttons on the forms page corresponding to which type of indexing you want (simple or fancy). Choose one or the other. The online

form can show you examples of each indexing format. Figure 1-4 shows you the file type icons used with fancy indexing.

Figure 1-4 Icons Used for Automatic Indexing

Where is your home page located?

When users first navigate to your server, they usually start with a URL like:

`http://www.[yourdomain].[dom]`

This displays your server's home page. To set your server's home page, you can do one of two things: create an index file in your document root, or specify here the name of a specific HTML file in the document root to use. If you do not wish to have a document root, or wish to keep your home page outside your document root, put the full pathname in the field. However, the home page can only exist on the server machine. A path to another machine won't work.

If you supply the name of a home page you already have, make sure the server system user has *read* and *execute* privileges to access it.

If you leave this field blank the server assumes you've created an index file or are using automatic indexing.

Submitting the Form

Once this form is all filled out and you're happy with your entries, click the "Submit this form" button. If everything you entered is valid, the next page you see says, "Success!" and you're presented with the opportunity to quit the install or continue on to the next form. If the entries weren't valid, you'll be asked to fill out the form over again.

Initial Administrative Configuration

The third form you see using the server installer sets up and controls the initial Netsite Communications Server administrative configuration. Answering the questions on this page, you'll:

- Assign an administrative user name
- Assign an administrative password
- Establish which hosts have administrative access to your server

The administrative user name field defaults to *admin*. You can use that or put your preferred administrative user name on the configuration worksheet. There are no defaults for the password field. Come up with something unique that is not a word in the dictionary and is undecipherable.

To get access to the administrative forms, use a URL similar to:

```
http://www.[yourdomain].[dom]/admin/
```

When you access your server's administrative forms, Netscape Navigator (or whatever network navigation software you're using) displays a dialog box that requests a user name and a password. When this happens, you'll need to give it the user name you enter here and the password you set here.

What administrative user name do you want to use?

You need to select and remember a user name for your administrative forms. We recommend the user *admin* but you can use any HTTP user name you want. The server will take care of creating the user for you.

What administrative password do you want to use?

You need to remember the password you give here. If you don't there is no way to find out later. If you review or print your configuration information, the password won't be appear there.

You'll need to confirm the password you entered on the form. If you don't supply a password in both fields, the form won't be valid when you submit it.

Which hosts do you want to allow administrative access?

Which IP addresses do you want to allow administrative access?

The first item can be filled out with either specific hostnames or a wildcard pattern. The second item can be filled out with either specific IP addresses or a wildcard pattern (see "How to Use Wildcard Patterns" on page 23 in this guide). Once the server is installed, you administer and manage it using Netscape Navigator. Obviously, you only want authorized people to administer your server. You need to indicate here which hosts are allowed administrative access. All others get an error if they attempt access.

If you do not trust the security of the network between other machines and your server, you should access the Netsite Server Manager only on the server machine itself so that administrative information never goes over the network.

A usable hostname can either include a full hostname such as:

```
www.[domain].[dom]
```

or a wildcard pattern designating a range of hosts. However, if DNS or your name service application isn't up and running properly, hostnames can't be used. Alternatively, you can specify hosts by their IP addresses instead,

using an IP address such as 198.93.92.103, or a wildcard pattern. Multiple hostnames or IP addresses must be separated by commas.

Note: Specifying nothing in these fields is equivalent to specifying a path of "*".

If you leave the answer to this question blank, anyone can attempt to login to the Netsite Server Manager. The administrative password keeps them out. If you answer the question with a single hostname, a wildcard pattern, or a series of hostnames, all other hosts which don't match the names or patterns can't get in.

Submitting the Form

Once this form is all filled out and you're happy with your entries, click the "Submit this form" button. If everything you entered is valid, the next page you see says, "Success!" and you're presented with the opportunity to quit the install or continue. If the entries weren't valid (which they won't be if you didn't enter an administrative password twice), you'll be asked to do the form over again.

View Your Installation Setup

Once the administrative forms are complete and submitted, you're offered the opportunity to view a summary of the installation entries you've made. If you want, once the information is on-screen, you can print the summary. It's probably a good idea to print it out and compare it with what you've written on your configuration worksheet.

Register Your Server with Netscape Communications Corporation

When you've answered the last question, submitted the online forms and reviewed your installation setup, you're offered an opportunity to register your server with the Netscape Communications Directory of Services. This is entirely optional, but if you do, you'll be automatically entered in our network directory. Either way, you can click the "continue" link and go to the final installation page.

Install the Server

If you're certain the server is configured to your satisfaction, then click the "Install your Netsite server!" link. The installer then sets up the directories and configuration files, and launches the server processes. You can then either visit your Netsite server as a user or access the Netsite Server Manager to administer it.

Chapter 2

Netsite Server Manager

Once you've completed the installation and your server is up and running, you may need to go back and either set up new items or reconfigure existing items as your system changes and grows. This chapter tells you how to access and use the Netsite Server Manager. The chapter also demonstrates how to use wildcards, how to automatically restart the server, and how to start and stop the server from the command line.

How to Access the Netsite Server Manager

Follow these steps to access the Netsite Server Manager using either Netscape Navigator or some other forms-capable network navigation software:

1. Launch Netscape Navigator or your forms-capable network navigation software.
2. Use the URL:

   `http://[servername].[yourdomain].[dom]/admin/`

   to display the admin login.
3. For the login name, use the user name you entered during setup and install.
4. For the password, use the administrative password you entered during setup and install.

Netscape Navigator or your forms-capable network navigation software displays the entry page of the Netsite Server Manager. The imagemap at the top of the page (shown in Figure 2-1) provides a navigation interface to quickly access the function you want.

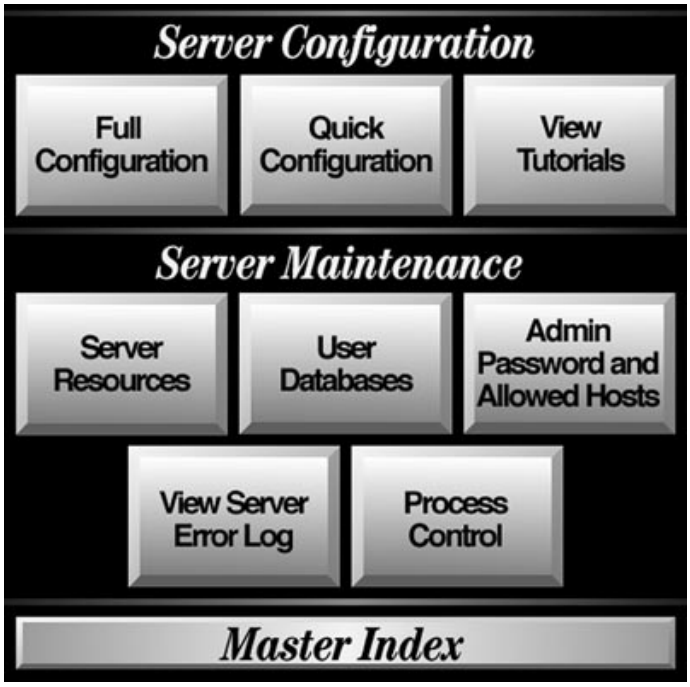


Figure 2-1 Netsite Server Manager Imagemap

Click a box on the imagemap to display the appropriate section or advance to the link you want and use it. If you need help, refer to the section in this guide that best suits what you're trying to do. The specifics of Server Configuration are covered in Chapter 3. Server Maintenance is covered in Chapter 4.

Many of the items that are available under Server Configuration and Server Maintenance are more directly accessible using the Master Index. You might look there first if you only have a single, quick administrative task to carry out.

How to Use Wildcard Patterns

To ease server setup, the server lets you specify certain items using wildcard patterns. By using wildcard patterns in some of your entries, you can cause any number of strings to match a single string.

A simple example of this functionality is shell wildcards. If you ask for **.html*, then *foo.html*, *bar.html*, and *baz.html*, all match the pattern, whereas *foo.bar* doesn't.

Inside a wildcard pattern, the characters in this list have special meaning. If you ever need to use one of these characters without the special meaning, precede it with a backslash (\).

`*`	matches zero or more characters
`?`	matches exactly one character and it can be any character
(*foo*`|`*bar*)	is an OR expression. It matches either the substring foo, or the substring bar. The substrings can contain other special characters such as * or `$`.
`$`	matches the end of the string. This is really only useful in OR expressions.
[*abc*]	matches one occurrence of the characters a, b, or c. Within these expressions, the only character that needs to be escaped in this is the bracket; all others are not special.
[*a-z*]	matches one occurrence of a character between a and z
[^*az*]	matches any character except a or z
~	followed by another shell expression removes any pattern matching the shell expression from the match list

Examples:

`*.sgi.com`	matches any string ending with .sgi.com
`(core|flop).sgi.com`	matches either core.sgi.com or flop.sgi.com
`198.93.9[23].???`	matches a numeric string starting with either 198.93.92 or 198.93.93, and ending with 3 digits

`*.*`	matches any string with a period in it
`*~foobar-*`	matches any string except those starting with foobar-

Restart the Server Automatically

Once installed, the Netsite server and its child processes run constantly, listening for and accepting requests. If your machine crashes or is taken offline, the Netsite server processes die with it. Make sure your server is configured for automatic restart on reboot with the following procedure

1. Use the *chkconfig* command to see if it is set to "on":

```
# chkconfig | grep netsite
```

2. If you see:

```
netsite off
```

Enter the following command:

```
# chkconfig netsite on
```

and repeat step 1 until you see:

```
netsite on
```

When the system is rebooted, the server starts automatically.

Start the Server Manually

If you should ever need to start the server from the command line, you must log in as root or become superuser and type this at the command-line prompt:

```
# /etc/init.d/netsite start
```

Stop the Server Manually

If you should ever need to stop the server manually, log in as root or become superuser, check the full process load using *ps -el* to see if other users might be using the server and, if not, type this at the command-line prompt:

```
# /etc/init.d/netsite stop
```

Chapter 3

Server Configuration

In the Server Configuration section of the Netsite Server Manager, you have three choices. You can:

- Do a full configuration
- Do a quick configuration
- View tutorials

This chapter covers the full configuration and briefly describes how to use the quick configuration. The tutorials are covered in Appendix B.

Full Configuration

This section shows you how to configure various technical aspects of your server. In this section, you can learn how to:

- Configure technical aspects of your server, such as its name or its port number
- Change your server's document root
- Change your server's home page
- Change your directory indexing preferences
- Create a directory dedicated to running CGI programs
- Create a template for user's public directories
- Set the default MIME type for your server
- Create a pointer to a resource that has moved

Note: The server allows you to create *resource templates* (see "How to Use Resource Templates" on page 104 in Appendix B) to ease maintenance of complex sites. With templates, you can apply the same configuration to several physical directories. You might not need to use them.

These are things you'll want to set up in your server. Occasionally, you'll need to reconfigure them as your server grows and changes. If you do not understand any of them, read about them here or follow the online links and read the various introductions.

You need to restart the server for any of your changes to actually take place. Once you submit the forms for any changes you make, you get a pointer to the script you should use to restart your Netsite server.

Configure the Specifics of Your Server

With this form you control various technical (non-document related) aspects of your server's operation. Through this form, you can:

- Set a server name
- Set a port name
- Set a server root
- Set a user for the server to run as
- Set where logging information is sent
- Set the maximum number of processes the server is allowed to use

Server Name

This field on the Technical Configuration form lets you change the name of your server. If you don't remember all the issues related to server names, review "Think of a Name for the Server and Set Up an Alias" on page 2 or "What do you want to name your server?" on page 9, both in Chapter 1. With the proper information handy, fill out this field on the form if you want to change the name of your Netsite server:

```
What do you want to name your server? [default]
```

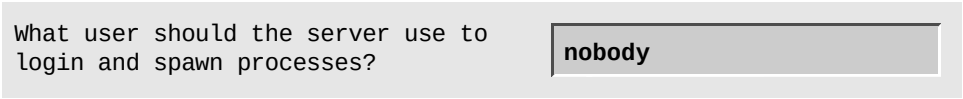
Server Port Number

This field on the Technical Configuration form lets you change the server port number. If you don't remember all the issues related to server port numbers, review "Choose a Unique Port Number" on page 4 or "What port number do you want to use?" on page 10, both in Chapter 1. With the proper information handy, fill out this field on the form if you want to change the port number of your Netsite server:

```
What port number do you want to use? 80
```

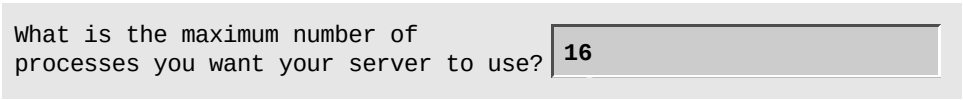
Server User

This field on the Technical Configuration form lets you change the user account the server uses to login and spawn processes. If you don't remember all the issues related to the server user, review "Create an IRIX User Account" on page 4 or "What user should the server use to log in and spawn processes?" on page 11, both in Chapter 1. With the proper information handy, fill out this field on the form if you want to change the server user for your Netsite server:



Server Processes

This field on the Technical Configuration form lets you change the maximum number of processes the server can launch. If you don't remember all the issues related to server processes, review "What is the maximum number of processes you want your server to use?" on page 12 in Chapter 1. With the proper information handy, fill out this field on the form if you want to change the number of server processes for your Netsite server:



Server Log Options

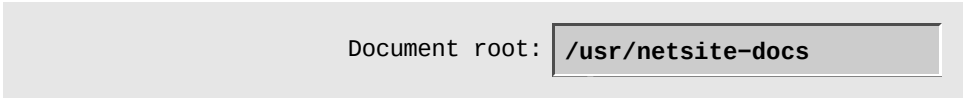
Under normal operation, the Netsite server maintains two log files in your server root directory (which you named earlier during setup and installation). One file logs server errors; the other logs who accesses your server. You can have errors reported through your system's *syslog* facility if you want, as indicated on this form. You can enable or disable access logging in the Server Maintenance forms, under Server Resources (see the reference online).

If you want to direct server logging to *syslog*, use the online form and click in the box to mark the option.

This is the last item in the online Technical Configuration form. You can review your changes by scrolling up and down the page. You can alter your changes by typing new information in any of the fields. You can clear your changes by clicking the “Clear this form” button. You can abandon all the changes on this form by clicking the “Return to configuration overview” link. To make your changes permanent, click the “Submit this form” button.

Change Your Document Root

This form lets you tell the server the new location of your server’s document root, if you’ve moved or changed it. If you don’t remember all the issues related to the document root, review “What directory do you want to use as the document root?” on page 14 in Chapter 1. With the proper information handy, fill out this field on the form if you want to change the document root for your Netsite server:



You can also tell the server not to use a document root for your server by leaving the field empty and then clicking in the box to mark that option.

You can review your changes by scrolling up and down the page. You can alter your changes by typing new information in the field or by checking or unchecking the box. You can clear your changes by clicking the “Reset to original” button. You can abandon all the changes on this form by clicking the “Return to configuration overview” link. To make your changes permanent, click the “Change document root” button.

Change Your Server's Home Page

This form lets you tell the server the new location of your home page, if you've moved or changed it. If you don't remember all the issues related to the home page, review "Make a Home Page" on page 2 or "Where is your home page located?" on page 16, both in Chapter 1. With the proper information handy, fill out this field on the form if you have changed the name or location of your home page:

Home page:

You can also tell the server to use an index file from the document root by clicking in the box to mark the option.

You can review your changes by scrolling up and down the page. You can alter your changes by typing new information in the field and by checking or unchecking the box. You can clear your changes by clicking the "Reset to original" button. You can abandon all the changes on this form by clicking the "Return to configuration overview" link. To make your changes permanent, click the "Change home page" button.

Change Your Server's Directory Indexing Preferences

This form allows you to change the style of directory indexing used by the server.

The Netsite server creates an index of directory contents automatically every time a directory is accessed that doesn't have an index file with one of the names you entered during setup and installation. These automatic indexes come in two flavors, simple and fancy. A simple index displays a list of the directory contents by name only. A fancy index also displays icons, file sizes, and last modification dates.

There are buttons on the forms page corresponding to which type of indexing you want (simple or fancy). Choose one or the other. Figure 1-4 shows you the file type icons used with fancy indexing.

Create a Directory Dedicated to Running CGI Programs

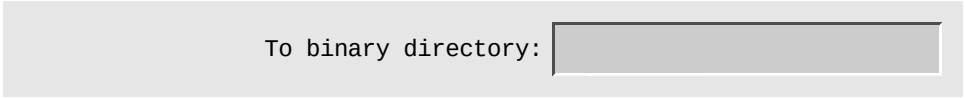
This form lets you designate certain directories as Common Gateway Interface (CGI) directories. If you don't use CGI programs, you can skip this form. In order to select a directory where CGI programs can be executed, use this form to enter a new URL prefix and then enter the name of the directory to which that URL points. If you want to restrict access to the CGI directory, you'll need to enter the names of hosts or IP addresses for which you want to allow access. If you don't want to restrict host access, you can leave the hosts and addresses entry fields blank.

Note: Imagemaps no longer require CGI to be set up. Refer to "What Is An Imagemap?" on page 91 in this guide to learn how to use ISMAP in the Netsite server.

When setting up the CGI external program interface, you can either leave CGI programs in the same directories with your other documents (and use the Server Maintenance forms to activate CGI as a file type in that directory), or you can designate a few central directories which hold the CGI programs. When you designate directories to hold them, you must create new URLs that point to them. Whenever a URL is navigated which begins with the prefix you designate on this form, a CGI program can be executed in the directory you've specified.

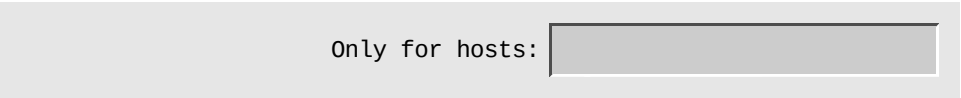
Add a New CGI Directory

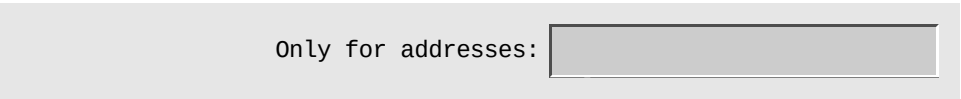
If you want to add a new CGI directory, use the online forms to fill out these items:



Restrict Host Access

If you wish, you can also make sure that only people from certain host names or from certain IP addresses can see your new URL. Any others will get a "not found" error. You can supply a wildcard expression to designate the allowable hosts. If you want to restrict access, fill out these two fields in the online form:





You can review your entries by scrolling up and down the page. Entries can be altered by typing new information in the fields. You can clear your entries by clicking the "Clear this form" button. All the entries on this form can be abandoned by clicking the "Return to configuration overview" link. To make your entries permanent, click the "Add this mapping" button.

You can also change any existing CGI directories you have by following the link at the bottom of the page.

Configure Users' Public Information Directories

This form lets you customize how you want users to create HTML accessible documents within their home directories. Through this form, you can:

- Decide on a method of specifying user directories
- Decide what is allowed within those directories

Users' public information directories are handy if you want to let your users make their own documents and home pages available. One way to do this is to create a central directory for your users, and keep subdirectories by each user name. You would then use the URL Management page to create a new

URL prefix for this directory (there's a link in the online form that lets you do this. Also see "Add a New URL Prefix" on page 37 in this guide).

Another way to enable user public information directories is to allow your users to create a subdirectory inside their home directory and place their documents there. If you choose to do this, the server automatically looks in your *password file* to find the user's home directory. Normally, the password file will be */etc/passwd*, or perhaps available through the Network Information Service (NIS). In this case, the server uses standard library calls to find a user's home directory. However, if you want to keep a separate password file, used only for the HTTP server, you can specify that file's pathname on this form.

You might also want to load the entire user database on startup. When you enable this option, the server scans the entire list of users and their home directories, and stores them in memory. The advantage of loading the user database at server startup is that lookups are very fast and it reduces network traffic if you're using NIS. The disadvantage is increased memory usage—but this shouldn't matter on most systems.

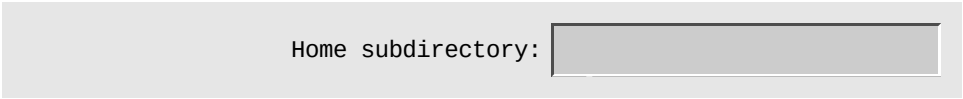
Most likely, you'll want to restrict the things users can do from their home directories. To do this, create a configuration template with your restrictions (see Appendix B, "How to Use Resource Templates" on page 104 for more information). If you do this, the name of the template should be specified on this form.

If you want to duplicate the default behavior of the NCSA httpd, use a value of /~ for the URL prefix, and *public_html* for the home subdirectory.

If you decide to allow user directories, follow these steps:

1. Enable user public directories by unchecking the "disable user public directories" box on this form.
2. You also need to choose a URL prefix to remap.
3. Then choose a subdirectory within the user's directory to look in.

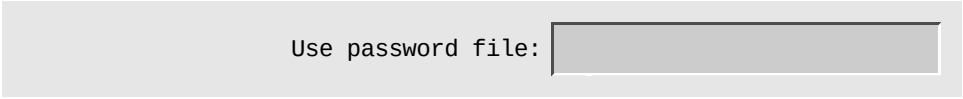
To enable user directories, fill out these two fields:



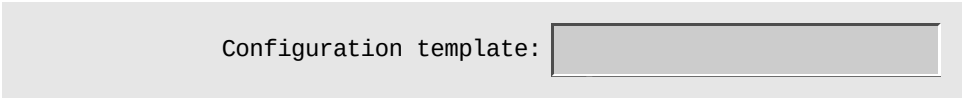
You can also tell the server to load the user database at startup by clicking in the box to mark the option.

To use the system password file, leave the button enabled.

To use an alternate password file, click the system password button off, then supply the name of a password file in this field:



To restrict user options using a configuration template, fill out this field with the name of the template to use:



You can review your entries by scrolling up and down the page. Entries can be altered by typing new information in the fields. You can clear your entries by clicking the “Reset to original” button. You can abandon all the entries on this form by clicking the “Return to configuration overview” link. To make your entries permanent, click the “Change” button.

Add a New URL Prefix

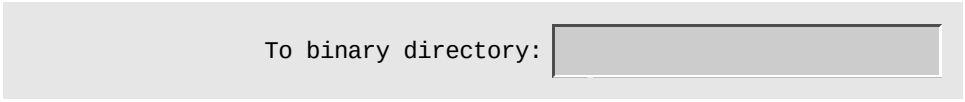
This form allows you to create a new URL mapping that points outside of the usual document root. Through this form, you can create a new URL mapping.

You need to specify what URL you want to map, and where you want it to map to. You can leave the other fields blank.

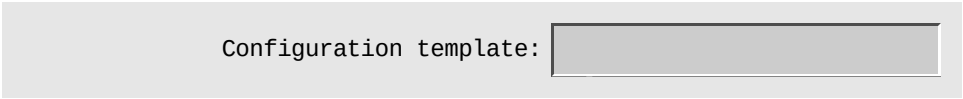
The server does not make your entire directory tree available to clients. Customarily, using the Netsite Communications Server, you designate a central directory (a *document root*) which contains your server's documents. However, sometimes you'll want to reference items outside this directory. To do this, you can either create a *symbolic link* (see *ln*(1) for information on how to do this), or you can create a new URL prefix here to map outside the document root.

When you create a new URL prefix, any directory with the same name in the document root will no longer be available. Any URL to your server starting with the prefix you map here gets its documents from the directory you specify on this form.

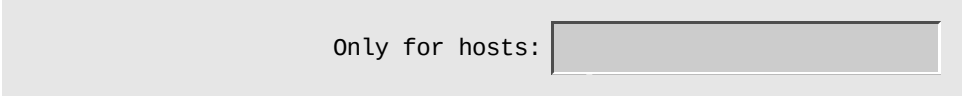
If you want to create a new URL mapping, fill out these fields:

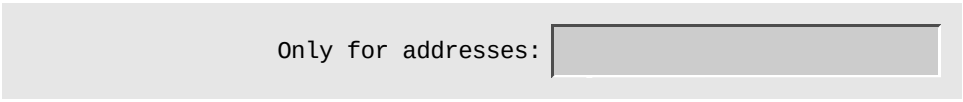


You have the option of making every request coming from this directory follow a certain configuration template. To do so, specify the name of the template in this field:



You can also make sure that only people from certain host names or from certain IP addresses can see your new URL by specifying the allowed hosts or IP addresses in these fields. Any others will get a “not found” error. You can use a wildcard expression to designate the allowable hosts or addresses.





You can review your entries by scrolling up and down the page. You can alter your entries by typing new information in the fields. You can clear your entries by clicking the “Clear this form” button. You can abandon all the entries on this form by clicking the “Return to configuration overview” link. To make your entries permanent, click the “Add this mapping” button.

You can also change any existing URL mappings you have by following the link at the bottom of the page.

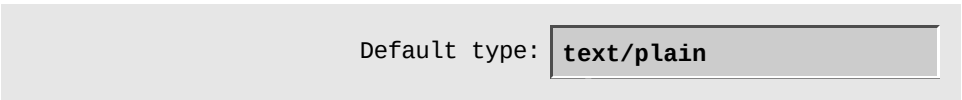
Change the Default MIME Type

This form allows you to change the default Multipurpose Internet Mail Extensions (MIME) type to send to clients. Through this form, you can specify a new MIME type to send when the server doesn’t know what a file is. You don’t have to change your default MIME type, but it may help the

Netscape Navigator correctly parse a file that doesn't have a recognizable extension.

The server uses MIME file types to distinguish various types of files, so that network navigation software like Netscape Navigator know the difference between an HTML file and a GIF. The server is required to give this information to the client.

The server usually does this through filename extensions. However, sometimes a file has no extension and the server can't tell what kind of file it is. When this happens, the server will send back a *default type* to the client. This default type is simply a guess. Most files without extensions are text files, but if this is not the case on your system, you should change this entry by entering a default MIME type in this field.



You can review your entry by scrolling up and down the page. You can alter your entry by typing new information in the field. You can clear your entry by clicking the "Reset to original" button. You can abandon the entry on this form by clicking the "Return to configuration overview" link. To make your entry permanent, click the "Use this default type" button.

Create a Pointer to a Moved Resource

This form allows you to redirect a client to a document's new home once the document has moved. Through this form, you create a pointer to redirect Netscape Navigator or other network navigation software to another server.

You need to fill out a prefix on this server that you want to redirect, and a URL to send the client to.

On occasion, you'll want to migrate information from the currently installed server to a new one. When you do this, you must give clients who have copies of the old URLs a pointer to the new resource. To do this, you can create a *redirection*. When a client receives a redirection from your server, it then looks at the new URL.

You must choose what kind of URL navigation string to send the client. If you want to send all requests for any document to another document, then you should select a *fixed URL*.

If you are moving an entire resource, and preserving its directory structure, you can make your URL a *URL prefix*, and when people access the old URL, the server will construct the new URL. For instance, if you move the */movies/* directory to a new server, and you give the URL prefix:

```
http://newserver/stuff/movies/
```

the server will redirect a request for:

```
/movies/oldies.html
```

to the new location:

```
http://newserver/stuff/movies/oldies.html
```

If you choose this option, and the prefix is a directory, you should make sure that you are consistent with trailing slashes. That is, you should not supply the prefix */movies* and redirect to the URL prefix

```
http://newserver/stuff/movies/
```

To redirect clients to another server, enter the new URL in this field:

Then choose one of the two button options (fixed URL or URL prefix) and supply the new URL in this field:

You can review your entries by scrolling up and down the page. Entries can be altered by typing new information in the fields. You can clear your entries by clicking the “Clear this form” button. All the entries on this form can be

abandoned by clicking the “Return to configuration overview” link. To make your entries permanent, click the “Add this mapping” button.

You can also change any existing redirections you have by following the link at the bottom of the page.

Quick Configuration

This form allows you to get a “Configuration at-a-glance” overview of how you have personalized your server. If you are still unfamiliar with the way the variables should be set, you might want to follow the other link to the fully documented version instead. The field items here are virtually identical to those described earlier in this section. If you know what you are doing, feel free to go ahead. Otherwise, use the fully documented version.

Chapter 4

Server Security

The Netsite Communications Server provides advanced security features which make HTTP communications secure. These features ensure the privacy of communications between Netscape Navigator and Netsite Communications Servers and allow the identity of the Commerce Servers to be verified. This chapter helps you understand some of the concepts behind private and authenticated server communications and then goes on to provide installation instructions. A final section discusses the issues related to maintaining server security. If you know all about privacy, encryption and authentication, you can skip the first part of this chapter and just read the part about security installation.

Understanding Privacy & Authentication

The Internet is a public network which interconnects millions of computers world wide. The Internet, the world's largest network, is used by an estimated 30 million people to transfer many forms of data from one computer to another, including: hypertext documents (for Netscape Navigator and other network navigation software), electronic mail, file transfers, and news from newsgroups.

Few Internet hosts are directly connected to one another. For data to move from one computer to another it almost always has to travel through several other computers. This traversal is called *routing*. The routing of sensitive data over the Internet is problematic because of two potential security problems: the inability to maintain privacy and the ability of third parties to illegally[1] pose as a principal in a conversation or transaction and make one of the parties believe it is the other. In the following example we have three

[1] 18 U.S.C. §§ 2510-2520 Electronic Communications Privacy Act (ECPA)

Internet players: Alice (a client), Mallet (a router) and Bob (a server). They are interconnected as shown in Figure 4-1

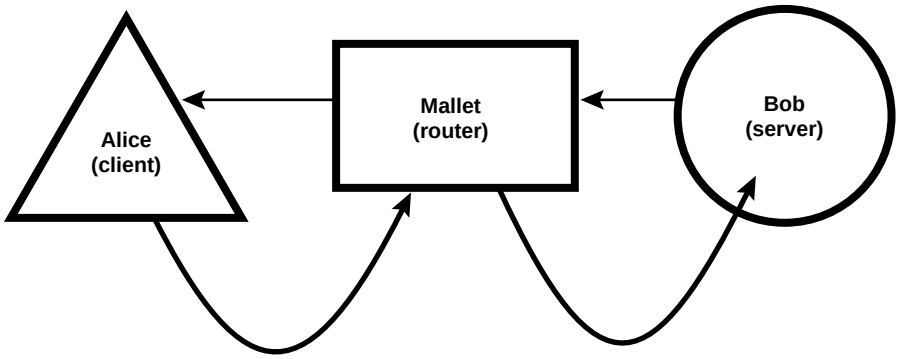


Figure 4-1 Routing Between Hosts

For Alice to communicate with Bob, Mallet must route data sent from Alice to Bob. Mallet can be a dedicated router or a general purpose computer. Without some kind of security mechanism, all data that passes between Alice and Bob can be seen by Mallet. If Mallet so desires he can make copies of everything that Alice and Bob communicate. In addition, it is possible for Mallet to pose as Bob simply by answering messages sent to Bob's network address. Alice has no way of verifying that messages received from Bob were actually sent by him. Unchecked, these problems with privacy and authenticity create the possibility of fraud and forgery on a broad scale.

How Netsite Security Works

Netscape Communications Corporation, working with RSA Data Security Inc., has developed security software that allows Netsite Communications Servers to transfer data over the Internet that is both private and authenticated. Netscape Communications has built patented RSA cryptographic technology into its online document navigation system. The Netsite Commerce Server and Netscape Navigator allow users to conduct commerce and secure communications over the Internet with ease and confidence.

Authentication

The Netscape Navigator and the Netsite Commerce server provide authentication using signed digital certificates. Signed certificates are issued by trusted third parties known as Certificate Authorities (CAs). The signed digital certificate contains two key pieces of information (see Figure 4-2). First is the certificate information itself, including the name of the key holder, their public key, the certificate's validity dates and the name of the certificate issuer. The second piece is the digital signature of the CA. The digital signature cannot be forged. It is encrypted using the private key of the CA.

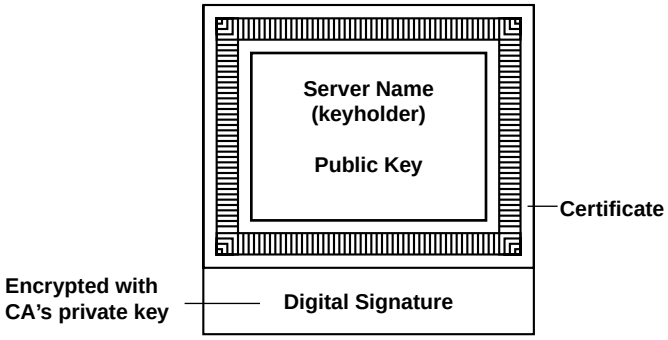


Figure 4-2 Contents of a Signed Certificate

Netscape Navigator validates a certificate using standard techniques. It looks up the public key of the CA using the issuer's name in the certificate. Once it has the public key, the client can compute a "message digest" of the certificate information (the name of the key holder, the public key, etc.). The digital signature contains a version of this "message digest" that has been encrypted with the certificate issuer's private key. The client uses the public key of the named certificate issuer to decrypt the signature and then compares the resulting message digest with the message digest it computed from the certificate body. The certificate is accepted as authentic only if both message digests are identical. Once this check has been made, the client challenges the server to prove it has the unique private key associated with the public key provided in the certificate. The server must pass the challenge for communications to continue.

Using the previous example, Bob has been issued a certificate by a trusted third party. By following the approach briefly explained above, Alice can verify Bob's certificate. If the certificate is found to be authentic, Alice knows she is conversing with the real Bob. Alice can then send Bob a "secret key" which she encrypts using Bob's public key that she received as part of Bob's certificate. Alice then knows that only the holder of Bob's private key can decode this secret key, also known as the *session key*.

Using certificates for server authentication, it is no longer possible for Mallet to pose as Bob to Alice. Mallet may indicate that he is Bob, but only Bob has the private key that goes with the certified public key. Therefore when Alice sends the session key to Mallet (who is attempting to pose as Bob), Mallet can't decode it or any subsequent encrypted messages from Alice. Mallet is foiled in his attempt to masquerade as Bob.

Privacy

Netscape Communications delivers privacy by encrypting data sent between two Internet hosts acting as client and server. A session key is exchanged which allows the client and server to encrypt and decrypt data, preventing a third host from observing their communications. Netscape Navigator and the Netsite Commerce server support a variety of cryptographic algorithms from RSA, including RC2 and RC4. The session key is exchanged using RSA's patented public key cryptosystem, which allows the session key itself to be exchanged securely.

Figure 4-3 Authentication & Encryption Model

Returning to our example (refer to Figure 4-3), Alice connects to Bob A, a server, and Bob B sends Alice a certificate which includes his public key. After validating Bob's certificate C, Alice generates a session key D and uses Bob's public key to encrypt it. Upon receipt, Bob decrypts the session key E using his private key. The two can then communicate securely using the session key. The session key remains valid over multiple connections until either the client (Alice) or the server (Bob) chooses to end the session. Were Mallet attempting to listen in, he would be unable to decrypt the message because he doesn't have the session key, which was originally encrypted using Bob's public key. Alice and Bob's communications are therefore private.

Security Installation

The procedure for installing security on your Netsite Communications Server is simple and straightforward. You use a form to generate a private and public key pair, and then send email to a certification authority requesting a digitally signed certificate. When the certificate is transmitted back to you from the certification authority, you install it using another form and your server can now encrypt data.

To begin this procedure access Server Security from the Netsite Server Manager by either clicking on the Secure Configuration button on the imagemap or by following the link to "Configure security for your server."

Configure Security on Your Server

The first page you see describes the process of setting up security on your server and provides a link to background information similar to what you're reading here. To actually start setting up security on your server you need to follow three basic steps:

1. Generate a key pair file, and send a certificate request to a certificate authority.

   In this step, you designate a key filename, a key file password and identify your server as you want it to be seen in transactions. Then you email a request to a certificate authority for a certificate that uses a

digital signature to uniquely identify your server. Clicking the Submit this form button on the form generates the key pair and sends the email to the certificate authority.

Note: Keep in mind that you must wait for the actual certificate to arrive in email from the certificate authority before proceeding to the next step.

2. Extract the certificate from a mail file.

   After you receive your server's certificate from the certificate authority, you need to decode the message and tell the server where to install the certificate.

   Note that this requires you to save the email message you receive from the certificate authority, and tell the server where to find it.

3. Activate security.

   Finally, you tell the server where to find the key file and the certificate file. When you kill and restart the server (from the command line), security will be active.

Server Key and Certificate Request

This form is accessed using the "Generate a key pair file" link from the "Configure Security on Your Server" page. The Netsite Commerce Server requires two special files to operate properly: a key file and a certificate file. Through this form, you can generate the key file, and send a request to a certificate authority for a certificate.

Before You Start

Your secure server requires two special files to be created before security can be enabled:

1. Key pair file

   The key pair file holds the public and private keys that your Netsite Commerce Server will use during secure communications. The private key data in the file is stored in encrypted form using a password you will specify below.

2. Certificate file

The certificate file contains an ISO-compliant signed certificate. A certificate is a digitally signed document that binds a name to a public key. When you fill out the name fields below you are defining the name of your server. The public key information will be automatically provided during the key generation step.

You can't generate a certificate yourself. Only a certificate authority can issue certificates. This form generates a certificate request which is emailed to a certificate authority.

Key Filename

The key pair that is generated by this form is saved in a specific file. The usual place to store these is in the server root, in the directory *[server root]/admin/config.*[1]

Remember where you put this file.

Type the filename in this field in the online form.

```
ServerKey.der
```

Key File Password

This is where you type in a password that will be used to protect your key pair and restart your Netsite Commerce Server. Any time your secure server is restarted this password is required so that it can decrypt the key file and extract the public and private keys.

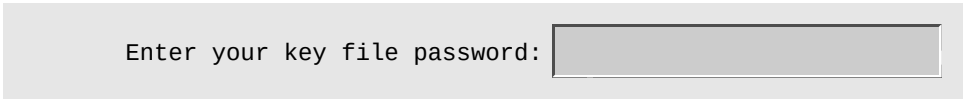
This password must not be forgotten.

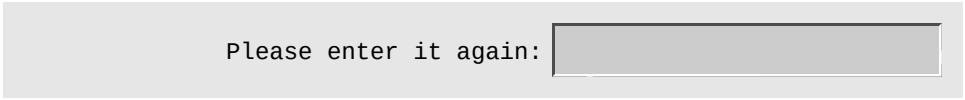
If the password must be recorded, then the physical security of the recording is your responsibility.

---

[1] The Netsite Server Manager automatically fills in this path for you. It is the default path to your server root appended with "/admin/config".

Because of the importance of this password, you need to enter it twice to make sure there are no typing errors. The password must be 8 characters in length. It is strongly recommended that the password have at least one numeric character somewhere in the middle. The password should not be a real word and should not exist in any dictionary of any language.

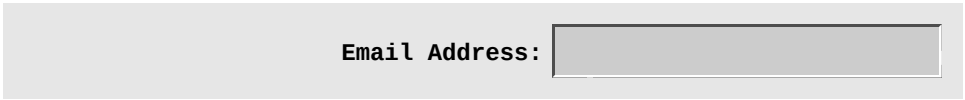


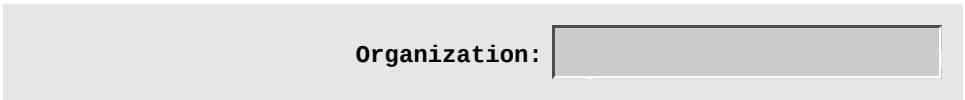


Server Name

By completing this section you define the name of the server that will be stored in the certificate. You should enter your name, your organization (company), and where you are located.

Items in **bold** are required; the other values are optional.





Organizational Unit:

Locality:

State or Province:

Country:

Please provide a telephone number where you can be reached. A certificate will not be granted unless other evidence regarding your identity can be provided.

Telephone Number:

Certificate Authority

Enter the email address of the certificate authority you wish to use. If you don't know what to do here, use the default.

authority@mcom.com

Finishing Up

Pressing the “Submit this form” button causes this form to generate a new key pair and store it in the file you chose above. The key pair will be encrypted using the password you chose above. Next, a certificate request will be generated and emailed to the certificate authority specified above. The name in the request will be constructed from the entries made above.

Please double check everything before submitting.

This is the last item in the online Server Key and Certificate Request form. You can review your changes by scrolling up and down the page. You can alter your changes by typing new information in any of the fields. You can clear your changes by clicking the “Reset this form” button. You can abandon all the changes on this form by clicking the “Return to the security home page” or the “Return to the admin home page” link. To make your changes permanent, click the “Submit this form” button.

You’ll have to wait awhile before you receive the certificate in email and can go on to the next step. Now might be a good time to create some documents. Check out the URL:

`http://home.mcom.com/home/how-to-create-web-services.html`

Decode the Certificate

When you get your email back from the CA with your certificate, you need to use a form to decode the certificate. This form is accessed using the “Extract the certificate from a mail file” link from the “Configure Security on Your Server” page. Save the email from the CA into a file on your machine. It is recommended that you save the file into your server root’s */admin/config* directory using the name *ServerCert.txt*. The information in the email is encrypted but you should take every protection to prevent unauthorized access to the certificate file (see “Protecting the Private Key” on page 61).

Locating the Text

Once you have saved the email from the Certificate Authority into a file, enter either the full pathname or the relative pathname from *[server root]/admin/config* in this field on the form:

```
ServerCert.txt
```

Specify destination directory

The server needs to know where to install your Netsite Commerce Server certificate. The recommended place is in the server root under */admin/config*. You can enter the name as a relative path from *[server root]/admin/config* or as an absolute path.

Note: Remember where you put this file as well**.**

```
Put the certificate in:
```

This is the last item in the online Decode the Certificate form. You can review your changes by scrolling up and down the page. You can alter your changes by typing new information in any of the fields. You can clear your changes by clicking the “Restore default values” button. You can abandon all the changes on this form by clicking the “Return to the security home page” or the “Return to the admin home page” link. To make your changes permanent, click the “Submit this form” button.

Basic Security Setup

This form is accessed using the “Activate security” link from the “Configure Security on Your Server” page. This page lets you configure your basic security setup. Through this page, you can:

- Enable or disable server security
- Specify a location for your server key file
- Specify a location for your server certificate
- Change the port that you will be running on

Enable/Disable Security

Most of the time, you want your server to run with security enabled. You may, at other times, want to disable security. If you temporarily disable security for some amount of time, make sure you return to this page and enable security prior to processing transactions which require privacy and authentication.

You only need to click a single checkbox on this page to enable security for this server (see the form online).

Specify the Key File

The server needs to know where to find your key file. You can specify it in this field in the online form in a relative path from *[server root]/admin/config* or as an absolute path.

Specify the Server Certificate

The server also needs to know where to find your certificate. You can specify it in this field in the online form in a relative path from *[server root]/admin/config* or as an absolute path.

Change the server's port

The default HTTP port number is 80, but the default HTTPS port is 443. If you enable security, you should change this value to 443. If you have security disabled, this value should be 80.

If you are not running on the standard ports, you can leave this value as is. For more information about using non-standard ports, see "Choose a Unique Port Number" on page 4, earlier in this guide.

This is the last item in the online Basic Security Setup form. You can review your changes by scrolling up and down the page. You can alter your changes by typing new information in any of the fields. You can clear your changes by clicking the "Restore default values" button. You can abandon all the changes on this form by clicking the "Return to the security home page" or the "Return to the admin home page" link. To make your changes permanent, click the "Submit this form" button. Then restart your server. Security will be enabled when the server resumes operation.

Now That Your Server Is Secure

You need to keep a few things in mind once security is enabled for your Netsite Communications Server is secure. Secure URLs are constructed using *http***s** instead of simply *http*. You need to understand how documents are accessed with a secure server document root and how those accesses are logged in the secure server log. You may need to know how to run a Netsite Communications Server (not secure) in the same environment. You also need to understand how HTTP access authorization works in a secure server setting.

Secure URL Construction

URLs which point to documents on a secure Netsite Commerce Server must now be constructed as:

```
https://[servername].[yourdomain].[dom]
```

Note that the customary *http://* is replaced with *https://*. All URLs with *https://* in them point to secure servers.

Secure Server Document Root & Logging

Once security is installed and enabled on a Netsite Communications Server all communications between the server and Netscape Navigator are private and authenticated. This means that any document served to a Netscape Navigator user from your document root is automatically encrypted. There is no way around this. Other client software may not work with a Netsite Communications Server. Make sure your users are using Netscape Navigator network navigation software from Netscape Communications Corporation.

The Secure Log

Once security is enabled, a new log, called “secure,” 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 the level of security. The bigger key size, the better security. The Netsite Communications Server uses a key length of 40.

Unprotected Server Document Root

If you want to serve documents from a document root that does not have security features (such as the Netsite Communications Server or the NCSA prototype httpd), it's highly recommended that you operate such a server on a different machine. If your resources are limited and you must run an unprotected server on the same machine as your Netsite Communications Server follow these guidelines.

Port Number Assignments

Make sure that the secure server and the unprotected server are assigned different and unique port numbers. If your Netsite Communications Server is going to use the HTTPS standard port for encrypted transmissions (443), the unprotected server ***must*** have a different port assignment than that. Likewise if you plan to run the unprotected server on the HTTP standard port 80, your Netsite Communications Server ***must*** use a different port number than 80. Don't use the same port number and don't get the two confused!

Use CHROOT on the Document Root

Also, the unprotected server ***must*** have references to its document root redirected using the *chroot* command. See the online forms for more information about how to do this.

HTTP Access Authorization with a Secure Server

Once security is enabled on your Netsite Communications Server, what has been referred to as simple HTTP access authorization now becomes a more powerful tool for controlling access to your server. With a secure Netsite Communications Server, HTTP usernames and passwords are no longer sent in the clear over the network. They are now encrypted and therefore

private. See the related sections in this manual on how to set up User Databases and control access using HTTP access authorization (see “Access Control” on page 71).

Changes to the magnus.conf File

With a secure server installed, you should know about the following changes to the magnus.conf file (the server’s main configuration file). These new directives are briefly described below:

Security

Name and Description

The Security directive tells the server whether security is enabled or disabled.

Syntax

`Security` `on|off`

`on|off` is an either/or value. **Security on** enables security; **Security off** disables security.

ServerKey

Name and Description

The ServerKey directive tells the server where the server’s key file is located.

Syntax

`ServerKey` *keyfile*

keyfile is the server’s key file, specified as a relative path from the server root or as an absolute path.

ServerCert

Name and Description

The ServerCert directive tells the server where the server's certificate file is located.

Syntax

`ServerCert` *certfile*

certfile is the server's certificate file, specified as a relative path from the server root or as an absolute path.

Physical Security Issues

There's more to running a secure Netsite Communications Server than just generating key pairs and installing a certificate. You need to make sure the workstation that runs the Netsite Communications Server is physically secure, secure from Internet intrusion, and keep your passwords and the private key safe from theft, perusal or copying.

Lock The Door

In the best of all possible worlds no one would ever try to get at your workstation and compromise its integrity. This ain't the best of all possible worlds. So the best place for your Netsite Communications Server workstation is inside a room that can be locked and physically secured when it is not occupied by authorized personnel. And speaking of authorized personnel, make sure ***you are*** and that you know who else has access to the room.

Internet Security

Make sure you take all normal precautions against intrusion by various and nefarious Internet bandits and cruisers who don't have your best interests at heart. There's a lot of literature about how to do this and if you're a savvy system administrator, you've "been there, done that." If you're not up to

snuff on sniffing out snoopers, please take a look at our Further Reading section at the end of this chapter.

Your system is vulnerable to attack from the moment you turn it on and connect it to the Internet. Some of the common security holes you should take measure to protect include:

- password cracking (see the next section)
- anonymous ftp
- port mapping in NFS and NIS (yp)
- telnet and rlogin access
- running X-windows
- viruses and other programmed threats
- misuse of uucp

Some suggested measures you can take to protect your system include:

- Remove all processes you don't need or know about.
- Exercise tight control over what processes are launched from inittab and rc scripts.
- Don't allow access via telnet and rlogin.
- Limit the number of actual Unix user accounts on the server machine.
- Study everything you can get your hands on regarding computer security and Unix security in particular.

How To Handle Passwords

This is the terrible, ugly part of the whole thing. You need a password to launch the server after the security features have been enabled. You need a password to access the Netsite Server Manager. You probably have a passel of other passwords you need to do other things on your machine. And you keep all of those passwords in your wallet or purse on sticky notes, right? Not a good idea.

Passwords need to be protected like no other object in your life. Ideally, they should never be written down. If you must write it down, never associate it

with your login and do everything you can to make it look like something on a shopping list (i.e., include it in a list of other items, only one of which is the actual password). If you must write it down, obscure it by adding a few characters that only you know are not part of the actual password. For example, if your password was **ap4lux**, you could write it in a list as follows:

```
2 grapefruits
4 apples
2 lightbulbs
2 bars of luxury soap
```

This way you'd know (or remember) that **ap4** comes from the "4 apples" item and **lux** comes from the "luxury soap" item, and only you would know how to interpret the items on the list.

Passwords must ***not*** be any whole word that is known in any language, made up of words that are known jargon or can be looked up in any dictionary! Passwords shouldn't be made up of contiguous characters on a keyboard. They should always include a numeric character or punctuation characters mixed in with random alphabetic characters.

Protecting the Private Key

Every possible file protection measure must be taken to protect the private key from theft, perusal or copying. Were the private key stolen (i.e., from a backup tape, your disk drive or any other place it might be stored), all privacy and authentication would be compromised. Lose the private key and you've lost your server security completely. Take care that the location of the private key is never discovered by anyone you don't trust.

Further Reading

To begin learning and understanding issues related to security on Unix machines, start with:

Practical Unix Security. Simson Garfinkel and Gene Spafford. O'Reilly and Associates, 1992.

Online, you can point Netscape Navigator at:

```
http://www.cis.ohio-state.edu:80/hypertext/faq/usenet/\
security-faq/faq.html
```

Usenet newsgroups that regularly discuss computer security include: comp.security.misc; comp.security.unix; and alt.security.

For an incredibly dense but thorough look at all the issues related to privacy and authentication cryptography techniques, try:

Applied Cryptography: Protocols, Algorithms, and Source Code in C. Bruce Schneier. John Wiley & Sons, Inc., 1994.

For a backgrounder on related legal issues, check out:

Cyberspace and the Law: Your Rights and Duties in the On-line World. Edward A. Cavazos and Gavino Morin. The MIT Press, 1994.

Chapter 5

Server Maintenance

In the Server Maintenance section of the Netsite Server Manager, you have several choices. You can configure:

- Server resources
- User databases
- Administrative password and allowed hosts

You can also use the Server Maintenance section to:

- View the server error log
- Administer process control

You access these items from the Netsite Server Manager home page.

Server Resources

On the Netsite Server Manager home page, click the Server Resources item on the imagemap or use the “Configure server resources” link under Server Maintenance. This action displays the “Choosing a resource to modify” page where you use various buttons, links or fields to select a resource to modify. A server resource can be a file, a directory or the entire server itself. You can also choose a template to modify. The resource selection options shown on this page include:

- Choose an individual file
- Choose a directory
- Select a previously configured resource (has a popup menu of server resources you’ve already modified)
- Enter a wildcard pattern directly to modify
- Choose the entire server
- Make a new template

Once you’ve selected a resource to modify and clicked the corresponding link, the “Modifying a resource” page is displayed where you can configure or reconfigure different aspects of the selected resource. You can choose one of these resource modification items:

- Activate CGI as a file type in this resource
- Add a custom signature when serving files in this resource
- Choose index files for this resource
- Customize error response within this resource
- Customize request logging in this resource
- Customize server-side includes in this resource
- Restrict access to this resource through HTTP access authorization
- Restrict the usage of file system links
- Restrict which hosts are allowed to access this resource
- Set a default query handler in this resource
- Remove your changes to this resource

Each of these items represent links. You may have to scroll up or down on the page to find the specific link. The following sections in this guide show you how to access and use each of the resource modification forms.

You use the resource modification items here primarily to control or log access to a resource in some way. For example, you might want to protect a directory so that only certain hosts can access it, or you might want to password protect an important file.

The fields displayed in the online forms match the specific resource being modified and vary somewhat by resource, so they are not shown here in the guide. The information you need to exercise resource control is fully explained here and in the online forms.

You'll need to restart the server after you've modified any resource.

Activate CGI as a File Type

This form is accessed using the "Activate CGI as a file type in this resource" link from the "Modifying a resource" page. The form lets you keep CGI scripts in directories with other documents. Through this form, you can activate CGI as a file type in the directory you have selected.

Requirements and Options

You only need to click a single checkbox on this page to activate CGI as a file type in this resource. Then use the buttons to activate or reset your changes.

Description

The Netsite server allows you to use external programs through the CGI interface. There are two ways you can set up CGI on your server.

The first way to do this is to allow CGI programs to co-exist with your regular documents in the same directory. In this way, the server knows that *.html* files are HTML text, *.gif* files are GIF images, and *.cgi* files are CGI programs to be executed. This setup is very flexible, the only stipulation being that you should take measures to protect your C source code from access. You should also only activate CGI as a file type in directories where

you want it to be used (i.e. you might not want your users' public information directories to have CGI activation enabled).

The second way to set this up is to keep all CGI programs separate from your normal documents in a separate directory. Then you can use Server Configuration to tell the server which directories in your file system contain CGI programs. This method of CGI activation requires some additional administrative overhead in that you must explicitly add new directories to the server. However, this method does allow you to maintain some control over where the programs are executed.

Text Trailers

This form is accessed using the "Add a custom signature when serving files in this resource" link from the "Modifying a resource" page. This form allows you to create a trailer which is automatically appended to the documents in this resource.

Requirements and Options

If you decide to make a custom trailer, you need to fill in the types of files to which you want the trailer appended (the default is *text/html*). You need to select a time format from the "Select a format" pop-up menu or enter a custom format using the same format as the *strftime(3C)*[1] system call. You then need to type in the actual trailer that you want to appear at the bottom of the documents selected in this resource. Then use the buttons to activate or reset your changes.

Description

Often, you might want to append a trailer, or a signature, to documents from a certain directory. You can have this trailer contain the author's name (and a link to his or her home page), the date of the page's last modification, or any copyright information for the page.

[1] See your IRIX system's reference pages for the exact format.

The server can automatically determine the date of last modification for you. You need to specify the format and where in the trailer it is to appear. If you specify a custom format, you must use the format used by the *strftime* system call. In your HTML trailer, use *:LASTMOD:* where you want the last modification date to appear.

Directory Indexes

This form is accessed using the “Choose index files for this resource” link from the “Modifying a resource” page. The form allows you to specify the name of the index file you want to use in a directory.

Requirements and Options

You do not have to change the default index file names, but if you do, you only need to specify the new names. Then use the buttons to activate or reset your changes.

Description

When you provide a reference to a directory on your server, usually you want to have a file inside that gives people a list of the directory contents. When people follow a URL which points to the directory, the server sends back this file to give them a catalog of what’s inside.

You should choose a name to use for this purpose. The defaults already in the field are *index.html* and *home.html*. If you want to use more than one name, separate them with commas. When the resource is accessed, the server sends back the first index file it finds.

Error Processing

This form is accessed using the “Customize error response within this resource” link from the “Modifying a resource” page. The form allows you to change how the server deals with error conditions for the resource you have selected.

Requirements and Options

You don’t have to change the default error responses, but if you choose to, you need to specify the name of a file to send for each error type. Any file you name here should already be written and exist at the location you specify (use an absolute path). Use the buttons to activate or reset your changes. If you need to change the error processing for another error type, there’s a link on the “Success” page for you to edit the error responses again.

Description

On occasion, an error will occur. The client may ask to retrieve something that doesn’t exist, or that it doesn’t have permission to get. The client might ask for a file which the server doesn’t have permission to read (due to administrator oversight or something else). Or, the server can encounter an internal error (such as running out of machine resources) which prevents it from carrying out a request.

Ordinarily, the server puts together a canned response and sends it back to the client. This message is generic and not always very helpful to the user. Because of this, the server allows you, as the administrator, to specify an alternate file to send.

What the default errors mean:

- Unauthorized—an area protected by HTTP access authorization was accessed and the user failed to provide a valid user name and password.
- Forbidden—the server was unable to read the requested file, probably due to file system permissions.
- Not found—the client asked for something which doesn’t exist, or asked for something which is hidden from its view by IP / domain protection.

- Server Error—the server ran out of resources, tried to execute a CGI program which failed to produce output, or found a syntax error or missing parameter in a configuration file.

Request Logging

This form is accessed using the “Customize request logging in this resource” link from the “Modifying a resource” page. With this form, you can customize request logging for the selected resource.

Requirements and Options

You do not have to customize the default logging, but if you decide to, you should fill out the items you want to change. You can click a button that turns off logging for this resource. If you choose to enable logging for this resource, you can specify a file to use. You can exclude logging for specific hostnames or specific IP addresses or use wildcards in those fields. A checkbox exists to have the server log only IP addresses and exclude hostnames. Use the buttons to activate or reset your changes.

Description

The server can record various information about each request a client makes. This information includes:

- Host name
- The document requested
- Whether the transfer was successful
- How many bytes were sent in the response
- Whether or not the user was HTTP-authenticated

For compatibility with existing log file analyzers, the server uses the *common logfile format*, which is the format used by the CERN and NCSA HTTP daemons.

Note: You can change which resource you are editing using a link online, and thus make a separate log file for different directories on your server.

Sometimes you will not want certain hosts to show up in your log file. A good example would be hosts from your local organization. In this case, you can tell the server not to log requests from certain IP addresses or hostnames. You can specify those with a wildcard pattern, if you want.

You can decide not to log the domain names of connecting hosts and only get the IP addresses. This can save a bit of network traffic and system resources, but won't really impact server response time since the server does not perform this lookup while the client is connected.

Server-Side Includes

This form is accessed using the "Customize server-side includes in this resource" link from the "Modifying a resource" page. The form allows you to customize server-parsed HTML, also known as "server-side includes." Through this form, you can:

- Activate or deactivate server-parsed HTML
- Choose how the server decides what is server-parsed HTML

Requirements and Options

On this form, you need to toggle one of three radio buttons to tell the server what kind of activation is allowed, and toggle one of another three radio buttons to tell the server how to determine which HTML files to parse. Then use the buttons to activate or reset your changes.

Description

Normally, HTML is sent back to the client exactly as it is on the disk, with no server intervention. However, sometimes you might find it useful to have the server parse these files, and insert request-specific information or files into the document.

By default, any file whose name ends with *.shtml* will be treated as parsed HTML and parsed by the server. Files that end with *.html* are not parsed. If you don't want to use a different extension for parsed HTML, you can choose one of two options:

- Parse *every* HTML file on the server

- To avoid the performance penalty of parsing every single HTML file as it is sent to the clients, NCSA httpd allows you to enable the *exec (x)* bit of the HTML file. Then, any file which has the bit enabled is parsed. Any file which doesn't have the *exec* bit enabled won't be parsed. We do not encourage the use of this function, but provide it to maintain compatibility with NCSA httpd.

Access Control

Note: HTTP user / password authorization transmits user names and passwords in the clear. This makes it vulnerable to people who watch packets on the network. You should not rely upon HTTP access authorization as your only security procedure.

This form is accessed using the "Restrict access to this resource through HTTP access authorization" link from the "Modifying a resource" page. The form shows you how to restrict access to a resource through HTTP access authorization. Through this form, you can:

- Activate HTTP access authorization for the selected resource
- Select a database of users
- Select which users should be able to access this resource

Requirements and Options

You need to enter the name of a user database to use, as you entered it when you created it (see "User Databases" on page 75 in this guide to find out how to create user databases). Then you need to enter the name of a realm so that users can identify the part of the server to which they are connecting. Last, you need to enter a wildcard pattern to allow access for a range of users matching the pattern. There's also a checkbox on the form that lets you turn off access control for the selected resource. Use the buttons at the bottom of the form to activate or reset your changes.

Description

The server allows you to restrict the access to certain areas for clients who enter their user name and password when prompted. You probably don't want to apply authorization to your entire server. So you should first make

sure you've selected a specific resource to which you wish to apply access control.

A user database must exist for access control to work. You can create the database either before or after you set up access control for the selected resource. If no database exists, a server error will be returned when users attempt to access the selected resource.

The realm is the name the server sends back to the client so that users can identify which part of the server they're connecting to. For example, the realm in the administration database is "Server Administration." If you were controlling access to a music archive, you could call the realm, "Music Library." The realm is completely arbitrary and up to you to come up with a name.

Within the user database you can use a wildcard pattern to specify which users are allowed access.

Limit File System Links

This form is accessed using the "Restrict the usage of file system links" link from the "Modifying a resource" page. The form lets you disable file system links (a method of having a file appear in two places at once) in the resource you have chosen.

Requirements and Options

The default is to allow links in the selected resource. If you want to disable links, fill out the "From directory" parameter and then set the checkboxes and buttons according to how you want to restrict links. Then use the buttons at the bottom of the form to activate or reset your changes.

Description

In some directories, you may want to make absolutely sure that only files within that directory are served, and no others. The best example of this is users' public information directories. For example, you may not trust your users to exclude links to copyrighted data in their pages.

The server is only able to read files it has permission to see. Therefore, the server can't read anything that your users don't have permissions for already. The limitations provided here are only designed to make it harder, not impossible, for them to give that data away.

The "From directory" parameter tells the server where to start looking for links. If this is an absolute path, the server looks in any portion of the path following the one you enter. If this is not an absolute path, the server tries to find the first instance of a directory with the given name within the current path, and start looking there. A good example of a non-absolute path would be if you set *public_html* to be the home subdirectory of your user public information directories.

The server only does the link searches when it is serving from the selected resource. Nevertheless, even with a resource set, you should enter a path in the "From directory" field to avoid needless searches.

If you want to prevent your users from giving away all of your valuable information but still would like to allow them to keep links to their own information, you can tell the server to disable only those symbolic links for which the owner of the link is not the same as the owner (user) of the file or directory to which the link has been established.

Deny Existence

This form is accessed using the "Restrict which hosts are allowed to access this resource" link from the "Modifying a resource" page. The form allows you to configure a resource so that it is accessible to only certain hosts or IP addresses. The server then tells any others that the selected resource does not exist.

Requirements and Options

Note: Specifying no path is equivalent to specifying a path of "*".

You can leave the "path to protect" entry blank, but you should definitely enter specific hostname patterns and IP address patterns for the clients that you want to allow. You can use the default "Send Not Found" if you like or you can specify a file to send in response. Use the buttons to activate or reset

your changes. You can also use a link to edit any existing mappings that are already in place.

Description

There are two uses for this command: you can use it to hide certain files (like EMACS backup files or files whose names start with a dot) from all of your clients, or you can restrict the access of some materials to certain clients.

When deciding whether to deny a request or not, the server first checks the host restrictions and, if it passes, doesn't check the IP restrictions. If it fails the host check, it goes on to the IP restriction check. If it fails again, then the request is denied.

If you want to restrict access to some documents for certain clients, you can put your protected documents in a separate directory, then use URL Management (see "Deny the Existence of Certain Paths" on page 175) to create a new URL for this directory which is visible only by the desired clients. Any non-allowed client will get a `Not Found` error when they try to access the URL. This can provide security through obscurity.

Alternately, if you want to keep your protected documents along with the other documents, you can tell the server to protect these paths and allow only users from the given IP addresses or hostnames to access them. For example, if the resource you selected was *usr/www/resources*, leaving the path blank and submitting the form would automatically protect that directory. Similarly, you could protect all the files associated with a special printing project by choosing the resource *people/gutenberg* and leave the path blank.

Using a wildcard pattern, you could protect all your GIF files by putting *usr/www/*.gif* in the "From directory" path.

When access fails, you can specify that the server should deny the existence of the file, or have it send back an HTML file which tells the client that access has been denied.

Query Handling

This form is accessed using the “Set a default query handler in this resource” link from the “Modifying a resource” page. The form allows you to specify a default query handler for the resource you have selected.

Requirements and Options

You need to specify a full path to the program you want to use as a query handler, or check the box to use no default query handler for this resource. Use the buttons to activate or reset your changes.

Description

Sometimes, you might want to place ISINDEX tags in your documents. In this case, there has to be a server side program which handles these search queries. The Netsite server allows you to designate a single CGI program to handle search queries for all queries within the selected resource.

Remove Changes

Accessing this choice automatically removes all your changes and prompts you to select another resource to work on.

User Databases

On the Netsite Server Manager home page, click the User Databases item on the imagemap or use the “Configure user databases” link under Server Maintenance. This action displays the User Databases page where you can use links that allow you to set up databases that store user information or control user information in the databases. Through this form, you can:

- Create a new database
- Convert an NCSA style database to the new format
- Modify the administrative password of an existing database
- Add, edit or remove users from an existing database
- Remove an existing database

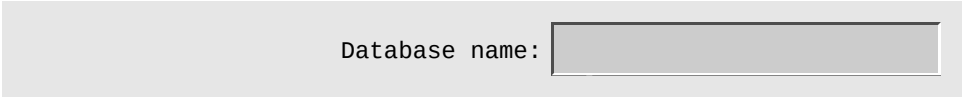
The Netsite server stores its databases in the server root, in the folder *admin/userdb*. When specifying a database, you should only use a name, not a full path.

Create a New Database

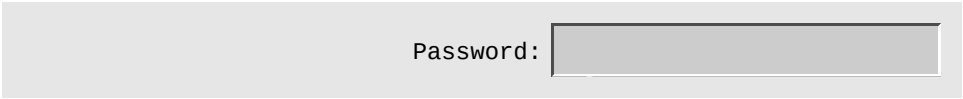
This form creates a new user database, stored in your server root in the directory *userdb*, which you can use with the HTTP authorization forms. Through this form, you can create a new user database.

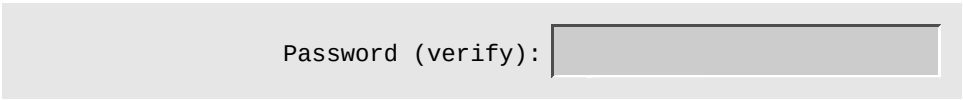
You need to specify a database name and an administrative password. There are links on this page that allow you to modify an existing database or convert an old NCSA httpd-style database to the new format.

First you need to decide on the database name. This is the name you'll use to refer to the database in the future, and it'll also be the file name used in the database directory.



You'll also need to set an administrative password. Enter it into both boxes, in case of a typo.





Then use the buttons at the bottom of the form to activate or reset your changes.

Convert an NCSA httpd-Style Database

This form allows you to convert user files similar to NCSA httpd's to the new format. Through this form, you can create a new user database to use in HTTP access authorization and set the administrative password for the new database

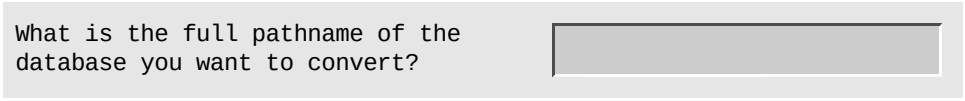
You only need to fill in a full pathname to the user file you want to convert, a name to give to the newly converted database, and the administrative password for that database.

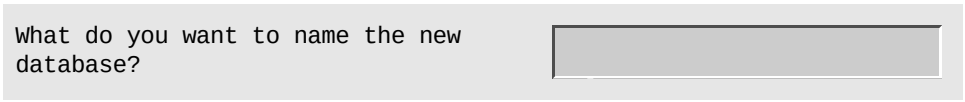
The format to be converted should look something like this:

```
user1:password1
user2:password2
user3:password3
user4:password4
```

Passwords are not normally stored in clear text. Usually, they are encrypted so that you cannot read them. If the file is an NCSA httpd file, the passwords will already be encrypted. Or, you can have the converter encrypt the passwords for you, if you check the appropriate box on this form.

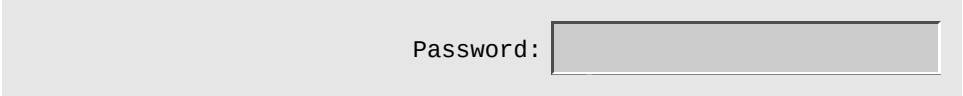
You then need to supply the full pathname of the database you wish to convert and name the new converted database.

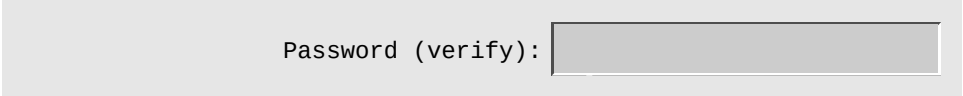




Remember, this is not the full path, just the name of the database.

You'll also need to set an administrative password. Enter it into both boxes, in case of a typo.





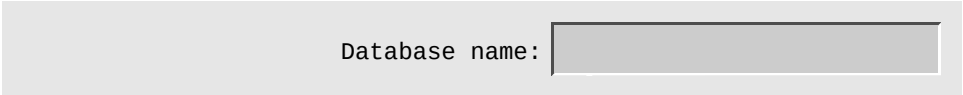
Then use the buttons at the bottom of the form to activate or reset your changes.

Database Administrative Password

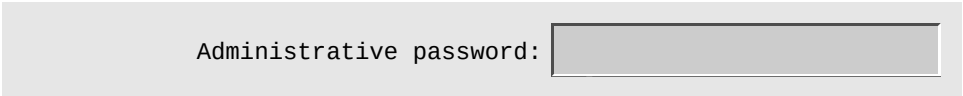
With this form you can modify a database's administrative password.

You need to fill in a database name, the current administrative password, and then the new administrative password.

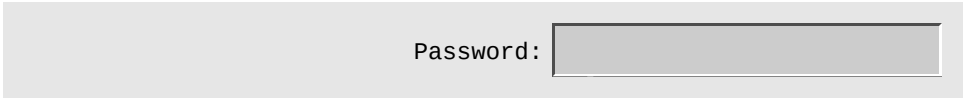
Enter the name of the database you want to edit. Remember, this is not the full path, just the name of the database.

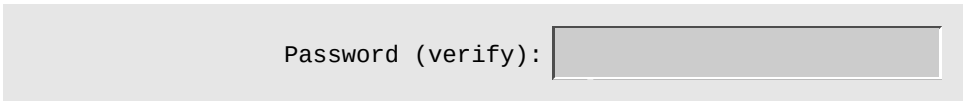


Enter the administrative password of the database you want to edit.



Then you can enter the new administrative password. Enter it into both boxes, in case of a typo.





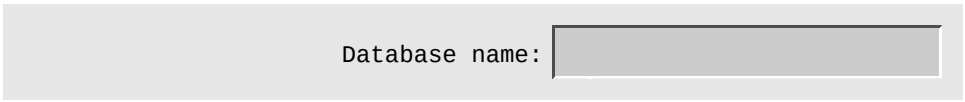
Then use the buttons at the bottom of the form to activate or reset your changes.

Add, Edit or Remove Users

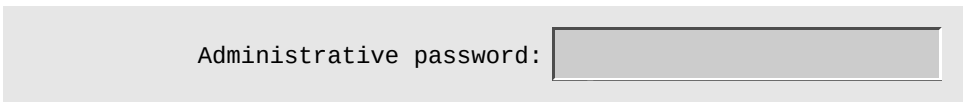
With this form you manage your database of users. Through this form, you can add, modify or delete users from this database.

You need to fill in a database name, an administrative password, and tell the server what action you want to take.

Enter the name of the database you want to edit. Remember, this is not the full path, just the name of the database.



Enter the administrative password of the database you want to edit.

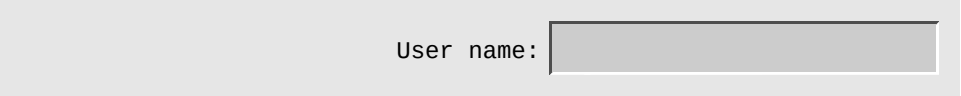


Action

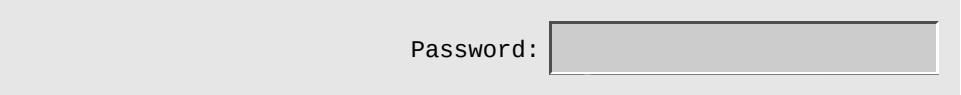
You can select the action to be taken here by activating one of three radio buttons:

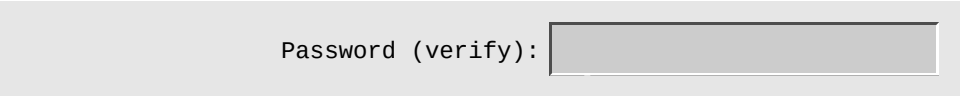
- Add user
- Edit user
- Remove user

Enter the name of the user you want to add, edit or remove in this field:



Enter another password only if adding or editing a user. You must enter it twice to make sure that there are no typos.





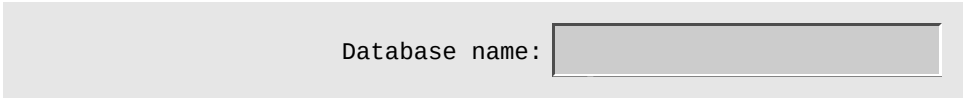
Then use the buttons at the bottom of the form to activate or reset your changes.

Remove a Database

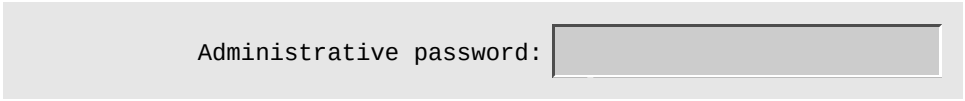
Through this form, you can delete a user database. Deletion is permanent, so choose wisely. Think twice before submitting this form.

You need to specify a database name and an administrative password.

Enter the name of the database you want to remove. Remember, this is not the full path, just the name of the database.



Enter the administrative password of the database you want to remove.



Then use the buttons at the bottom of the form to activate or reset your changes.

Administrative Access

On the Netsite Server Manager home page, click the Admin Passwords and Allowed Hosts item on the imagemap or use the “Configure admin passwords and allowed hosts” link under Server Maintenance. This action displays the Administrative Password form where you can use links that allow you to:

- Modify the hosts allowed to access the administration of your server
- Modify your administrative username
- Modify your administrative password

You only have to change the fields you want to modify.

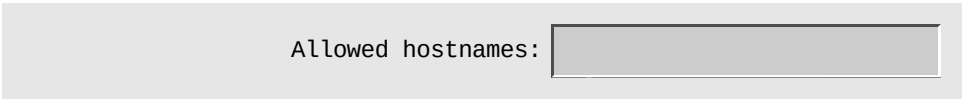
Note: Note that the IP address, 127.0.0.1, is added to the IP list regardless, in case of emergency.

You can specify allowable hosts by hostnames, IP addresses, or both, in wildcard patterns or comma-delimited lists.

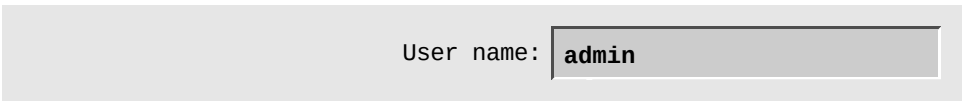
When doing the restriction check, the server first checks the host restrictions. If the check passes, the document is served. If the incoming request fails the check, the server then tries the IP restrictions. If they both fail, then the client is refused.

Here are some examples of restrictions:

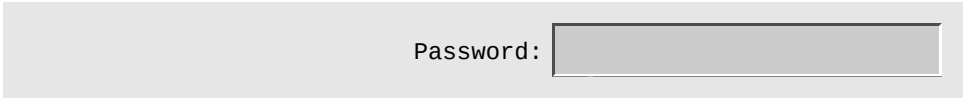
```
Hosts: *.sgi.com
Hosts: machine1.sgi.com,machine2.sgi.com
Hosts: machine*.sgi.com
Hosts: (machine1|machine2).sgi.com
IP: 192.82.208.*.
```

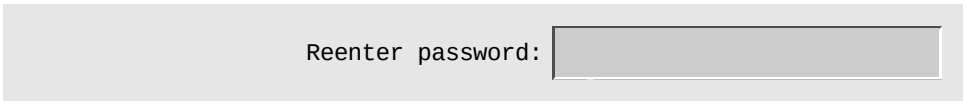


You can change your administrative username. *admin* is recommended, but not required.



You can also change your administrative password. Be sure to enter it twice, to make sure there were no typing errors.





Use the buttons at the bottom of the form to activate or reset your changes.

Server Error Logs

On the Netsite Server Manager home page, click the View Server Error Log item on the imagemap or use the “View the server error log” link under Server Maintenance. Clicking the link displays the most recent 25 lines in the error log. You can use a text viewer or editor to access the entire log by looking in the *logs* directory in the server root.

Process Control

On the Netsite Server Manager home page, click the Process Control item on the imagemap or use the “Process Control” link under Server Maintenance. Through this form, you can:

- Kill your server
- Soft restart your server
- Hard restart your server

Shut Down the Server

Sometimes, you might want to shut down your server, perhaps because you want to take it out of service while you do backups. Once you kill the server, you can restart it with */etc/init.d/netsite start*. Click the “Shut down the server” button to do so.

Soft Server Restart

After making changes to the administrative settings, you might need to do a soft restart of the server. This means that the server is sent a signal that tells it to re-read the configuration files and start again without interrupting service. Click the “Soft restart the server” button to do so.

Hard Server Restart

Sometimes, you might want to make sure the server is really fully restarted, or perhaps you changed a major variable like the maximum number of processes that requires that the server restart completely. Be warned, however, that doing a full restart might cause some users to get an error message if they try to access your server during the time when the server goes offline and before it comes back.

Note: Remember also that if your server is running on a port less than 1024, this script will not work. The full restart script cannot run as root, so if you're under 1024, you have to do it by hand. The Netsite server default port is 80, so unless you have changed that default, the full restart script will not work.

Click the “Fully restart the server” button to do so.

Appendix A

What's Different

The Netsite Communications Server is the first commercial HTTP server fully compatible with NCSA Mosaic and other HTTP clients and servers. Here you'll find descriptions of how the Netsite server compares to the NCSA research prototype server. If you've never used NCSA httpd, you may find this irrelevant—feel free to skip to the next part of the guide that interests you.

QuickStart Installation and Administration Interface

With the Netsite Communications Server, all server setup and maintenance (including out-of-the-box QuickStart installation) can now be performed through a set of HTTP-navigable document forms. This makes configuration and maintenance simple and fast. You can configure every feature of the Netsite Communications Server through the document forms interface.

With NCSA httpd, setup and maintenance of the server require you to directly edit configuration files. If you wish, you can still use a text editor to modify the configuration files directly (refer to Appendix C).

Reduced System Impact

The Netsite Communications Server uses a different process model than that used with the NCSA server. The Netsite server launches a set number of processes to handle all requests by clients accessing the server. When a request comes into the server, the kernel hands the request to any one of the available processes. After the request is complete, the process simply makes itself available again. This creates a group of processes that are always active and never destroyed. The Netsite Server Manager server allows you to limit the number of processes so you can manage the impact on your system. This mechanism reserves part of your machine's capacity for other uses and

prevents client requests from overwhelming system resources and crashing your machine.

NCSA httpd creates one new process for every connection requested by clients accessing your server. When a request is completed, its respective process is destroyed. NCSA httpd's inability to manage system load periodically causes busy servers to serve documents very slowly or crash often as system resources become exhausted.

Easier Imagemaps

The Netsite Communications Server allows you to set up images as imagemaps by referencing a map file (whose format is the same as map files used with NCSA httpd). When a user clicks on the image, the server automatically processes the request using the specified map file. No external program is necessary to handle the imagemap. Relative links can be used in map files.

To set up images as imagemaps using NCSA httpd, you must first edit and compile the imagemap program. Then, every time a user wants to create a map, you must edit at least two files. This can become difficult to administer.

Document Signatures, or Trailers

Only the Netsite Communications Server lets you create a custom signature, or trailer, that is automatically appended to the end of HTML documents without using server-parsed HTML. This signature can be changed on a per-directory basis, and can include such items as:

- The author's name (or contact address)
- Any copyright notice that applies to the pages in the directory
- A small form that allows you to search a database
- Anything else you want to include in every one of your documents

You can also have the server generate the date of last modification for each page and place it in the trailer (in a format you specify). For those of you who

use caching proxies, this function does not destroy the last modified date or content length headers.

This feature eliminates the extra system load caused by parsing each HTML page before including a custom trailer or signature. NCSA httpd can only complete this task using server-parsed HTML. For special situations and creative uses, the Netsite server still allows you to use server-parsed HTML. The Netsite server also repairs some of the problems present in the implementation of server-parsed HTML on other HTTP servers.

Custom Error Messages

When errors occur, such as the client asking for something that doesn't exist, the client asking for something that the server can't read or which it is not authorized to read, or the server encountering an internal problem and becoming unable to fulfill the request, both NCSA httpd and the Netsite server generate a precompiled error message to be sent to the client. With the Netsite server, you have the added flexibility of generating custom HTML error messages to send back to the client instead of the standard messages that are customarily built-in.

You can do this on a per-directory basis. For example, when users attempt to access a page for which they have no authorization, you can put contact information in the error message that tells them who to contact to obtain access.

Custom Logging

With the Netsite Communications Server you can log all accesses to a central file as well as log errors through the syslog facility. You can also add individual log files on a per-directory or per-template basis. With this feature, you can generate access logs for single directories or single documents. NCSA httpd reports errors to a central error log and accesses to a central access log.

Combined with access control, log information for a particular directory can be made publicly available without compromising the security of the complete, central log file. The Netsite Commerce Server also allows you to

choose whether to skip reverse DNS lookups when generating a log entry. Eliminating reverse DNS lookups for log purposes can save precious system resources.

Flexible Access Control

Access control is now more flexible, allowing cases in which local hosts don't require user and password identification, but remote hosts do. URLs can be effectively hidden from remote entities and the server can deny the existence of those URLs if unwanted / unallowed persons attempt to access them.

Improved User Management

The Netsite Communications Server provides a streamlined interface for setting up and managing HTTP access authorization. You can use the new Netsite administrative forms to manage user databases and manage users within those databases. Users can then use a forms interface to change their passwords. An optional new user database file format offers fast access to an arbitrary number of users.

Configuration by Directory or Template

With the Netsite Communications Server you can control every server function, both on a per-directory and by-template basis. When you create a template, you name a set of configuration parameters which can then be applied to multiple directories.

NCSA httpd only allows you to control access to the server and enable indexing on a per-directory basis. This makes controlling access to users' home directories difficult unless their directories are all in the same location.

Multiple User Public Information Directories

NCSA httpd allows your system's users (Unix users) to provide information from their home directories through a special URL starting with /~, such as

/~rob. The Netsite Communications Server not only allows you to do this but also allows you to change the user URL to something else, such as */u/* appended with the user ID (e.g., */u/rob*). If you want, you can also use password files other than */etc/passwd* or the NIS map so that you don't have to give your users' accounts on the server machine. This enables you to reduce traffic on your NIS server and reduce your server's overall load by telling the Netsite server to pre-scan the password file.

NCSA Features Not Supported

Due to a lack of interest or functionality that was improved elsewhere in the server operation, the following items are not supported:

- RFC 931 identity checking for logging purposes has been not been implemented.
- *.htaccess* files have no equivalent in the Netsite server. You can use the administrative forms to configure user directories.
- In favor of the CGI program interface, the antiquated NCSA script format (*/htbin*) is no longer supported.

For similar reasons, the following item has changed:

- Directory indexing can no longer be configured on a per-directory basis (but may be configured on a global basis).

Tutorials

These tutorials are intended to make installing your server a bit easier on you, whether you are migrating from NCSA httpd or simply installing the server for the first time. They'll help you with:

- Setting up image maps
- Setting up CGI directories
- Using HTTP user access control
- The difference between Unix users and HTTP users
- Making your server safe
- Optimizing your server's performance
- Using resource templates

What Is An Imagemap?

An imagemap is a "clickable" image that Netscape Navigator and other HTTP-based clients treat as a special case: clicking in different areas on the picture causes different URLs to be loaded. Graphical menus, icons, clickable building maps, and clickable blueprints are some of the many possible applications of imagemaps.

How Does This Compare To My Old Imagemaps?

In NCSA httpd, imagemaps were done by creating an *imagemap.conf* file in the server's configuration directory. This involves getting the site administrator to change a file for the user. Netsite has integrated imagemaps into a flexible and unified handling system that is easy to use, and is less work on the administrator.

How Does It Work?

Only two files are required for image mapping: a *.map* file and an actual image file. The *.map* file contains a list of coordinates to define the regions of the map that cause the different effects when clicked on.

How Are Regions Specified?

There are three different methods to specify the regions that can be clicked on: by circles, by polygons, or by simple rectangles. Each has the same basic structure…

```
method url coord_1 coord_2 ... coord_n
```

…each one uses the coordinates in a slightly different way. The coordinates are specified simply by listing an x,y pair. Here's how to specify the region types:

```
circle (url) center edgepoint
```

…a circle, specified by two coordinates: first, the center, and then, a coordinate on the circumference (any one).

```
poly (url) vertex_1 vertex_2 ... vertex_n
```

…a polygon, of up to 100 vertices. Each coordinate is a vertex. You do not have to close off your polygons.

```
rect (url) upper-left lower-right
```

…a rectangle, specified by its upper left corner and its lower right corner.

```
default (url)
```

…this is the URL that is loaded when none of the regions match.

Remember, the URL is the URL you want to have associated with the specified region. Also remember, you can use "#" as the first character in a line to specify a comment.

How Does It All Fit Together?

Once you have the *.map* file set up, you can put the imagemap into action. It can be referenced by the following structure:

```
<A HREF="myimage.map"><IMG SRC="myimage.gif" ISMAP></A>
```

The map file specifies the anchor, and the image file specifies the image to be displayed. *ISMAP* defines this as an imagemap and tells the server to watch for clicks.

Can You Give Me An Example?

Here is a sample *.map* file. We have an image, *mcom.gif*, which we want to make clickable. We have a circle, a rectangle, and a polygon in the shape of an M.

```
# mcom.map: Sample imagemap definition
#the circle around the logo
circle http://www.mcom.com/circle.html 25,25 0,25
#a rectangle inside the circle (one of many!)
rect http://www.mcom.com/rect.html 0,0 15,30
#the m in the middle
poly http://www.mcom.com/m.html 15,35 15,10 25,15 35,10
35,35 15,35
#anything else
default http://www.mcom.com/tryagain.html
```

Then the image is referenced by:

```
<A HREF="mcom.map"><IMG SRC="mcom.gif" ISMAP></A>
```

Methods of CGI Access

With Netsite, there are two methods of specifying where Common Gateway Interface (CGI) files can reside (See Appendix C for details). The first and most efficient of these methods is to simply activate CGI as a file type in a directory where *.cgi* files can reside along with other files (such as *.html*, *.gif*, etc.). The second method lets you keep your old NCSA httpd-style *cgi-bin* directory, and you just inform the server that the directory exists and to process the contents as CGI programs.

Activate .CGI as a File Type In Certain Directories

It is sometimes desirable to have only one CGI file among other non-CGI files, and to be able to tell the server that *.cgi* should cause those URLs to be treated as CGI programs.

How did it work before?

In *srm.conf*, you would first specify *.cgi* as a file extension mapping (SRM stands for Server Resource Map) . Then, in *access.conf*, you would activate CGI execution in the directory in which you wanted it.

How do I do it now?

Under Server Maintenance in the Netsite Server Manager there's a section called "Server Resources." When you access that section, you'll see several buttons, links and fields which let you select a specific resource to work on.

The first thing you want to do is select the directory in which you want to activate CGI execution. Use one of the links that lets you choose a directory or your document root. The Netsite Server Manager automatically displays a list of directories you can select. Click the directory in your document root where you're keeping your CGI programs. Now you are ready to activate CGI in that directory.

From the "Modifying a resource" page, click on this link:

- Activate CGI as a file type in this resource

Then mark the "Allow CGI programs in this resource" checkbox, and click the "Use this setting" button.

From the "Success" page click on the "Send the restart signal to the server" link or return to the Netsite Server Manager home page, click on Process Control, then click on "Soft server restart" and the changes will take effect.

Specify a CGI-BIN Alias

If you want to preserve the use of any existing CGI programs you may have in an old *cgi-bin* directory, you can choose that directory as a resource to modify, and then map a URL to that directory.

How did it work before?

NCSA httpd specified aliases in *srm.conf* through the *ScriptAlias* command, by which all the files in the directory specified would be interpreted as CGI programs, and executed accordingly.

How do I do it now?

Enter the Netsite Server Manager home page on your server, and click the Full Configuration item on the imagemap or choose the "Configure with full documentation" link. When that page is displayed, one of the items listed is:

- Create a directory dedicated to running CGI scripts

Click this link and you're presented with a form that allows you to specify the prefix to replace, the directory to map it to, and a method to allow only certain hosts to access the CGI directory. Enter the prefix you want to remap, the name of the directory where you want to keep your CGI programs, and what hosts should be allowed to see the directory.

For example, if you wanted to map:

`http://[yourhost].[yourdomain].[dom]/cgi-bin`

to */var/httpd/cgi-bin*, you would enter */cgi-bin* as the prefix to replace, and */var/httpd/cgi-bin* as the directory to map it to. If you don't care who accesses the directory, just leave the access restriction fields blank. If you want to restrict access, enter a wildcard pattern to specify what hosts and IP addresses can access your scripts.

Click "Add this Mapping." From the "Success" page, click the "Restart the Server" link or return to the Netsite Server Manager, click on Process Control and then click on "Soft server restart" to make your changes take effect.

HTTP User Access Control

Using Netsite's HTTP user access control system is easy, quick and intuitive. It uses a high speed database format to offer substantially improved performance for large database files. You can even convert NCSA httpd-style databases to the new system.

You might want to read this and look at the Netsite Server Manager pages while you're reading.

Adding user access control is a three-step process:

1. First, you should access User Databases under Server Maintenance to configure the user databases.
2. Then you can access Server Resources to choose the database to which you want to apply access control.
3. Then you can actually apply the access control using the appropriate link from the "Modifying a resource" page.

Create User Databases

The first step is to configure the user databases. You have two options when creating a new user database: you can create one from scratch, or you can convert an NCSA httpd-style database into the new format.

In either case, simply enter a name for the new database in the appropriate form field. All of the server's database files are kept in the server root, under *admin/userdb*. The name you enter is used as a file name for the database.

You also need to enter an administrative password for the database. A user named *admin* is created in the database. Remember what this password is—you'll need to enter it later.

If you're creating a new database, that's all you need. If you're converting a database, though, you'll also need to enter an absolute pathname in the appropriate field for the database to be converted. Note the format for the file, *user:password*, followed by a carriage return for each user.

The converter script can also act as a batch processor. Simply create an NCSA httpd-style user file with plain text passwords and click the checkbox to enable encrypting passwords. It will create the database, encrypting the passwords for you. That way, you can either type in this list yourself, or have a database program generate it for you (see “Convert an NCSA httpd-Style Database” on page 77 for the format of entries in the NCSA httpd-style databases).

Modify User Databases

You can access the User Databases section in Server Maintenance to modify user databases after you have created them. This allows you to add users, edit users, and remove users. Simply enter the name of the database in the appropriate form field, enter the admin password for that database, and make whatever changes you need.

Choose a Resource

Next, you must choose a resource to which you want to apply access control. Access Server Resources from the Netsite Server Manager home page to choose your user database as a resource. A resource is anything accessible by the server that it can return to a client; usually this means files and directories.

There are a number of ways to choose a resource: you can choose by file, by directory, or the entire server. Note that if you choose a directory, it will apply to the subdirectories as well.

Add Access Control

Once you have chosen the database as a resource, click on the “Restrict access to this resource through HTTP access authorization” link, and a form is displayed that lets you set up access control.

The database name is simply the name of the database you used earlier when you created it.

You can choose an allowed user pattern to specify who uses a resource. For example, the pattern *(mary | fred)* allows users "mary" or "fred" access to the database, but no one else. Entering no pattern will cause the server to allow everyone in the selected database to have access to that resource.

Don't forget to enter a *realm*, so that users know where they are connecting. For example, the realm on the server administration pages you used to navigate here is "Server Administration." The realm is completely arbitrary and it's up to you to come up with a name.

Now, all you need to do is click in the "Use no access control for this resource" box to unmark it.

Submit the form, restart the server, and the resource will be protected.

UNIX Users Versus HTTP Users

When you install your server, and when you use various features such as user public information directories and HTTP access authorization, you will be asked to create users. The term "user" can be misleading in this case.

The Netsite server maintains its own set of users, which are designated HTTP users because they only apply to Netsite. When you install the server, it asks you to select a user name and a password for the *admin* account. *This is not a real system account which people can use to log into the server machine.* It does not have a home directory, cannot read mail, and cannot do anything a normal user can do. It is only accessible through HTTP and the Netsite Communications Server. This user is used by the server to protect its documents from unwanted eyes.

The confusing point is when the server documentation discusses setting up "user public information directories" or asks what "user" you want to run the server process as. The user being referred to in this case is a Unix user, who can log in and has a home directory. Generally, the system keeps these users in a database called */etc/passwd* (although this database can be shared over the network using NIS).

Make Your Server Safe

It is easy to configure your server to the level of safety you want by configuring user directories, CGI, and sym-link usage. We call this *safety* instead of its traditional name of "security" to avoid confusion with the protection of certain documents through host restriction, HTTP user access control and the capabilities of the Netsite Commerce Server. See the earlier tutorial on "HTTP User Access Control" on page 96.

Making your server safer means making it more difficult for your internal users to compromise your system by accidentally leaving the door open for remote hackers to break into your system. To make it safer, however, you must restrict which server features your untrusted users can use.

User Directories

On some servers, such as a server with a large number of users, the administrator does not want to have too many people changing the document tree. In cases like this, it is possible to allow users to create their own public directories, and use name translation to allow the world to access them. In most cases, the user is allowed to create a directory called *public_html* in their home directory, and references to:

```
http://[yourserver].[yourdomain].[dom]/~user/file.html
```

will access:

```
~user/public_html/file.html
```

This function is activated through the users' public information directories configuration form.

However, this may not be appropriate for some servers. How much do you trust your users not to put inappropriate files in their public directories? If you trust them, it can be quite convenient and encouraging for the user to learn HTML (since they don't have to bother the administrator when they're doing it.) If you don't trust them, however, you may want to disable user public information directories in the configuration form.

If you trust your users not to put inappropriate content on your server, you probably still want to keep some restrictions on what is allowed. CGI

programs, along with symbolic links (known as *sym-links*), are a good example of something you may not want to allow in a public directory.

CGI Security Concerns

Netsite's support for CGI programs is an extraordinarily useful way for you to interface external programs to your server. Examples of CGI scripts include forms handlers, administrative document handlers, and on-the-fly document converters.

Sometimes the way CGI programs work, however, can raise security concerns, because they can be programmed to do something inappropriate. One concern is that the programs might be executed as the UID your server is running as. That user should not have privileges, but it will be able to read public files such as */etc/passwd*. Again, it is a case of how much you trust your users. If you keep your source code well protected and trust your document authors not to do anything inappropriate in the document root, you can safely activate CGI as a file type for the server. This allows you to put a CGI program in any directory, so long as its name ends in *.cgi* (see the earlier tutorial "Add Access Control" on page 97 and "Activate CGI as a File Type" on page 65).

On the other hand, you may want to keep an eye on the contents of the CGI programs, and keep them all in certain central directories. Traditionally, there has been a directory named *cgi-bin* in the server root which contains these programs. That way, you can protect the directory and control what is put in and out of it. While this means extra work for you, it does let you keep an eye on things if you don't trust your users.

Silicon Graphics does not recommend turning on CGI scripts in public directories. Although you may trust your normal users not to do anything unsafe, the possibility of abuse is too high to ignore, especially if your system is compromised.

You must strike a balance between trust and security.

Symbolic Links

Using symbolic links (or sym-links) is a way for the server to access files outside the document root without trouble. Although they can be done with name translation, sym-links don't require the user to bug the administrator every time a file outside the document root needs to be accessed. However, it also means that a user could create a sym-link that lets clients access files outside of the document root—possibly files that are inappropriate. The time-honored example is */etc/passwd* on a system which does not use shadow passwords. Another example could be the documentation to one of your licensed software packages—the user may mean well when he or she makes it accessible as navigable pages, but your licensing agreement may forbid making the copyrighted documentation public. The user might not be aware of this restriction.

There is another type of link, however: the non-symbolic or *hard link*. These links allow the user to create a pointer only to files on the same filesystem as the link. You may want to disable these kind of links as well.

Because of these concerns, the server gives you the option to control sym-links in one of three ways:

- No control at all
- Follow sym-links if owners match
- Disable sym-links

Both sym-links and hard links are not disabled by default.

Note: Disabling sym-links does not prevent the user from copying a file or directory which they can read.

"Follow sym-links if owners match" means that the server will only follow a link in the user's public information directory (or elsewhere) if whatever file or directory the symbolic link points to is owned by the same user as the target directory. That way, users can point links to their own documents without a problem but won't be able to point to documents owned by someone else (such as *root*). However, keep in mind that if a user really wants to make that file public, they may simply copy it to their directory, reducing the usefulness of sym-link security in the absence of disk space quotas. It's all a matter of trust.

"No control at all" is exactly what it implies: all sym-links are followed. This may be appropriate for a server with all trusted users, especially since it takes the server extra time to process a pathname looking for links.

Optimize Your Server's Performance

Once you have your server installed, you can optimize the server's performance to best fit your needs. On some servers which are not dedicated servers (that is, other people use the machine), you might want to keep its resource usage low. On the other hand, on a dedicated server expecting a great deal of requests at a time, you probably want to optimize it to offer the best possible response time.

There are two major factors affecting server performance: the number of processes it is allowed to run, and the usage of server-parsed HTML.

The Maximum Number of Processes

Setting the maximum number of server processes is done through either Full Configuration or Quick Configuration on the Netsite Server Manager home page. The number you choose can be anywhere from 1 to 1024. It should be a function both of what kind of load you expect and how much of a server load you can handle. On one hand, choosing too many processes could cause unnecessary memory swapping in your system as all of the processes take up the available RAM. On the other hand, choosing too few can cause delays when users attempt to access your server.

If your server is fairly low demand (under 10,000 accesses per day), or if your server is being used for other things, running 8 to 16 should be enough for your needs. If your server is higher demand, you may want to make sure you have enough RAM and set the number as high as you can. 32 processes should be enough for most high access sites, with 48 or 64 being even better if you have the RAM to spare.

To illustrate what happens if you choose a number of processes which is too low, suppose that you've chosen some number of processes for your machine. Suppose also that some number of people are currently connected to your server and downloading large files. When another client accesses

your server, that client is placed in a waiting queue and its request is delayed until another process becomes available. If you set the maximum number of processes way too low, then clients will begin to receive timeout errors after a certain number of them are in the queue.

Choosing the proper number of processes is mostly a matter of fine-tuning. The easiest way to tell how your server is doing is to try accessing it during peak hours to see what kind of response time you get. You can also use system-specific tools such as *top* to see how much memory your server is using.

Server-Parsed HTML

Normally, HTML is sent back to the client exactly as it is on the disk, with no server intervention. Sometimes, however, you might find it useful to have the server parse these files, and insert request-specific information or files into the document. This is known as server-parsed HTML, or server-side includes. The functionality originally appeared in NCSA httpd.

Server-parsed HTML is a very convenient function, but deciding which files to parse can be time consuming for your system and may cause an unnecessary load. By default, the server parses files with the *.shtml* extension. That way, when a request comes in, a simple check on the file name allows the server to decide whether or not to parse it. This is the method we recommend.

However, you may not want to use a different name for those files. You may want to keep parsed HTML alongside your non-parsed HTML and keep using the *.html* extension (and not have to rename the files you want parsed with the *.shtml* extension). In that case, you should use one of these options:

- Parse every document on the server
- Parse only documents with the *exec* bit turned on

Parsing every document can be a large performance hit for the server, especially where large documents are involved. If your server is fairly low-demand, or if you have a high performance machine running it, this may not be a problem. A good way to know if you have CPU resources to spare is to use a system-specific tool such as *top(1)* or *gr_osview(1)* to examine what percentage of your CPU time is idle during peak hours.

Because of this performance problem, however, the *exec* bit method was introduced. The *exec* bit method causes the server to only parse documents that have the *exec* bit set in their file permissions (i.e. mode 755, etc.) Although this is better than parsing all documents, many users have their *umask* set to have all of their documents executable. In that case, your server would be parsing documents that are not actually server-parsed HTML. *Netscape Communications Corporation considers this method of parsed HTML selection to be an error prone solution and does not recommend its use. The feature is provided for compatibility with NCSA httpd.*

Ultimately, you must choose the configuration that best suits your needs for convenience and performance. If you choose to use a method other than the default method for choosing what to parse, you can evaluate performance measurements with the parsing options enabled and with them disabled to see if they make a real impact in your environment. Most likely, you'll want to strike a balance between efficiency and compatibility.

How to Use Resource Templates

Resource templates are a fast and easy way to apply one configuration to a number of directories without having to reenter all the resource configuration information again. In essence, templates are named objects. That is, you create an object, configure it, then tell the server to use that configuration when accessing the directories you choose. Some example uses for resource templates are *cgi-bin* directories, public HTML directories, or perhaps a subscription directory scheme.

Create a New Template

Creating a new template is quick and easy. Access Server Resources from the Netsite Server Manager pages. At the bottom of the "Choosing a resource to modify" page there's an option for creating a new template. Activate the radio button titled "You can make a new template" then enter a name you want to use as a reference to the template in the field. For example, you might choose *my-CGI* for a CGI template. Then, just submit the form.

Modify That Template

You should now be in the “Modify a resource” section, with the template you selected as the resource. Make the changes you wish to make to the configuration.

For example, if you wanted this template to activate CGI, you might first select “Activate CGI as a file type in this resource.” Perhaps you also want to “Customize request logging,” and “Restrict the usage of file system links.”

Simply change the items you want to change, and when you are finished, submit the form, restart the server and return to the Netsite Server Manager home page.

Apply the New Template

Applying resource templates is done through name translation. With name translation, a virtual path is mapped to a physical path on the disk that may be quite different. For example, using the CGI example again, you might map:

```
/weather-CGI
```

to the directory:

```
/local/users/weatherman/CGI
```

The name translation page is often called the “URL prefix page” or “URL Management.” It is found at the bottom of the full or quick configuration pages, or it can be found quickly using the Master Index. First, choose the prefix you want to map to that directory. Then, select the physical directory to which you want to map it.

There is an item labeled “Configuration template.” Enter the name of your template here.

Submit the form, and presto, the configuration you entered for that template will now be applied to the directory you chose.

How About an Example?

Suppose Bob is running an opinions newspaper, and has several featured columnists. Bob wants the writers to be able to put their columns on the net without his intervention, but he doesn't want to give them access to the entire server. He decides to create a new template.

First, Bob creates the template, using "*Writers*" for the name. He then goes to the "Modify a resource" page and makes his changes. Bob decides to completely disable sym-links, make the index file *currentarticle.html*, and make a custom trailer that holds a number of things, including a disclaimer, a link to the server's index page, and the date the article was last modified.

After restarting the server, Bob adds his name translations. He maps:

```
/cynthia-brown
```

to:

```
/users/authors/cyndy/articles
```

Then he maps:

```
/bob-smith
```

to:

```
/users/authors/bsmith/editorials
```

and:

```
/swedish-chef
```

to:

```
/users/culinary/swedish/bork-bork-bork
```

making sure that he uses the *Writers* template for all of them. He restarts the server.

And now, Bob sits back and watches his writers fly!

Appendix C

CGI-A Primer

CGI stands for “Common Gateway Interface.” It is an interface between your Netsite server and external programs you write. CGI allows those programs to process HTML forms or other data coming from remote navigation software, and then send a response to the navigator which answers its query.

The word “gateway” can be interpreted in two ways. The original meaning of “gateway” describes a gateway to other services through the HTTP server. Some examples of other services are Archie / Prospero, WAIS, or a database of movie information. These services can’t currently be accessed natively through a World Wide Web navigator, but through a CGI program and an HTTP server, they can be accessed, and the results displayed in a rich hypermedia format.

Another way to view the term “gateway” is that CGI is a gateway between a program and your Netsite HTTP server. That is, CGI sits on the fence between your program and the server. In this appendix you’ll find out about:

- How CGI Works
- Accessing CGI Programs
- Information Provided by the Server
- Program Output
- An Example CGI Program in ANSI C
- Tips for CGI Program Development

How CGI Works

This section provides an overview of how CGI fits into the interaction of client software like Netscape Navigator and an HTTP server like Netsite.

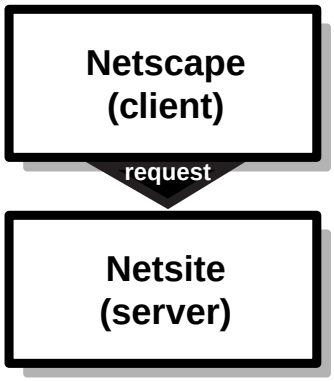


Figure C-1 Step 1: The Client Contacts The Server

The navigation software contacts the Netsite server running on your machine and sends it a request. This request may be for a document, or it may send the results of filling out an HTML form. If the server finds that the request is for a regular document, it sends that document back to the navigator. If it finds that the request is for a an external application, then the server uses CGI to execute that program.

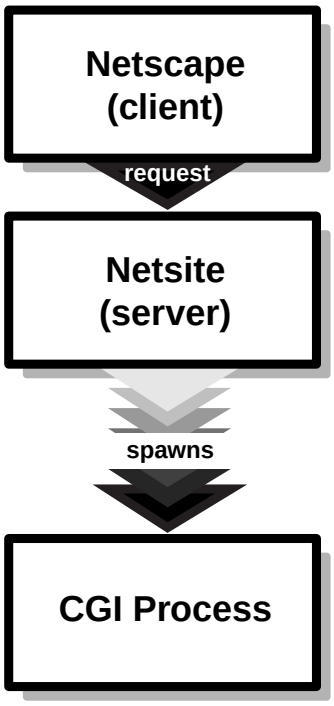


Figure C-2 Step 2: Create a CGI Process

When the CGI request comes in, the server creates a copy of itself, called a *CGI process*. This process has no purpose other than to set up communication between your CGI program and the server. Because it is a copy of the server, it has access to a great deal of information about the incoming request. Some examples of the kind of information to which your program has access can include things like which remote host is making the request, whether that host is using HTTP access authorization or not, what navigation software the remote user is running, and others. A complete list of these variables, referred to as "environment variables," is provided later in this appendix.
See "Variable Formats" on page 117.

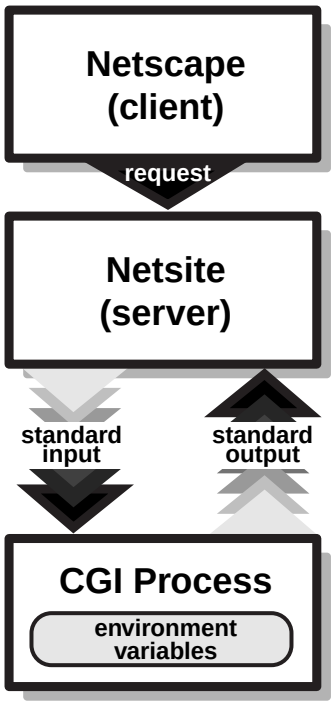


Figure C-3 Step 3: Assign Variables and Open Data Paths

The CGI process then takes the data the server has about the current request and encapsulates the data into these environment variables. Environment variables and file descriptors are the only forms of data which can be inherited by newly executed programs. They are used to give your program its data. The CGI process also creates data pathways between your CGI program and the server, so that the server can send it any encoded form data the client has submitted, and so you can send your reply back to the client via the server.

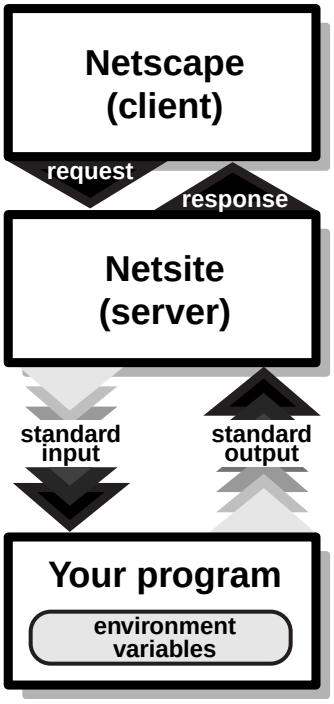


Figure C-4 Step 4: Execute the CGI Program

At this point, your program takes the data that the server provides, processes it, contacts any external services it needs to, and then sends its response to the server via the standard output. The server then takes the program's response, prepends any necessary protocol headers to the output, and sends it back to the client software.

Accessing CGI Programs

To determine if a URL the client is accessing refers to a CGI program, the Netsite server can use two mechanisms. Neither mechanism is active by default.

The first method is to designate files of a certain MIME type as CGI programs, and then allow those programs to reside in the same directory

along with your other HTML documents and graphics. This method is the most flexible, but you should be careful not to activate CGI in directories which you don't trust (such as directories to which users can upload or those that "untrusted" users own). When this feature is active, any file whose name ends with *.cgi* will be executed as a CGI program (see "Activate CGI as a File Type" on page 65).

The second method is a bit more restrictive, in that it allows only CGI programs to be run from certain specified directories. This gives you more control over who writes CGI scripts and who has access to where they're kept. By using this method, you designate certain virtual paths (or URL prefixes) as CGI directories. Then, any file inside those directories is executed as a CGI program. A popular example is the default virtual prefix */cgi-bin*, which maps to a directory in the server root called *cgi-bin* which contains all of your CGI programs.

If an attempt is made to access a CGI program, and you get a server error accompanied by the message `no way to service request for /foo/bar.cgi` in your error log, or the text of your program appears in the client's window, then you have not properly activated CGI in that directory.

Embedding Information in URLs

The method of CGI activation you choose determines only part of the URL used to access your program. URLs to CGI programs can be split into three different parts:

```
[virtual path][extra path information]?[query string]
```

The virtual path is similar to a path you would use to access a regular document or image. That is, it points the server to the file that contains the CGI program you want executed.

Extra path information is additional information you can embed in the URL after the program name. Extra path information is optional. This mechanism can be used for two purposes. First, it can be used to convey constant information to your scripts independent of the client's intervention. Second, it can be used to access the server's virtual-to-physical path translation mechanism. If you place another virtual path in this part of the URL, and your script needs to access a file outside of itself, you don't have to embed

filesystem paths inside of your CGI URLs. You can use other partial URLs instead. The server provides the physical pathname corresponding to that virtual path in the environment variables using path translation (see “Add a New URL Prefix” on page 37).

The query string is another optional part of the URL. It can either be explicitly given in your hypertext anchor, it can come from a user typing into a search dialog box for an HTML document with the ISINDEX tag, or it can come from HTML forms (see “Accepting User Input from URLs and Other Sources” on page 114).

Here are some examples of CGI URLs. In all of these examples, the script name used is */misc/search.cgi*, and the document root referred to is in the physical directory */d/netsite/docs*.

Example A:

```
http://yourserver/misc/search.cgi
```

This is a simple URL to the CGI program located at */d/netsite/docs/misc/search.cgi*, with no extra path information and no search query.

Example B:

```
http://yourserver/misc/search.cgi/type=minimal
```

In this example, the script author has embedded the extra path information */type=minimal* into the URL. In this case, the script author likely ignores the translated path information. The script can then read the raw extra path information, and perform a different kind of search based on that information. The same script could then be used with different path information to perform a different type of search.

Example C:

```
http://yourserver/misc/search.cgi/misc/movies.mdb
```

This URL specifies the extra path information */misc/movies.mdb* for this script. The search script could then use the translated path information to find the database */d/netsite/docs/misc/movies.mdb* and then search it.

Example D:

```
http://yourserver/misc/search.cgi?netscape
```

If this were a hyperlink, one use would be to create a link which automatically searches for the word *netscape* without the user having to type anything.

Example E:

```
http://yourserver/misc/search.cgi/misc/movies.mdb?netscape
```

This hyperlink would automatically return the results of a search for netscape in the movies database */d/netsite/docs/misc/movies.mdb*.

Accepting User Input from URLs and Other Sources

There are three cases in which the client software can send you information that the remote user inputs:

- HTML form data
- Input to an ISINDEX search dialog
- Clickable images (ISMAP or imagemaps)

In the third case, the data you receive will be given as a query string of the form *xx*,*yy* where *xx* and *yy* are the number of pixels from the upper left hand corner of the image the user clicked on.

When the data is typed into a search dialog or form text entry, the data will be encoded using URL encoding. In this encoding, there are two rules:

- Spaces are changed into plus signs
- Any of the characters can be “escaped” by changing them into a sequence of the form %*xx*, where *x* is a hexadecimal digit. The character is identified by translating the two hexadecimal digits into a number from 0 to 255, which is a character.

When the data is coming from a search dialog resulting from an ISINDEX tag, the above translations can be applied directly. Further, your CGI

program can receive this information fully translated on the command line if you want to avoid the hassle of performing this translation yourself.

If your data is coming from an HTML form, then the location of this data varies depending on the method attribute specified with the FORM tag in your HTML document. If the GET method is used, this information comes from the QUERY_STRING variable (see “QUERY_STRING” on page 120). If the POST method is used, this information is sent to your program using the standard input.

Wherever the data is provided, it will be of the form:

```
name1=value1&name2=value2 ... &nameN=valueN
```

If there are any equals signs (=) or ampersands (&) in the encoded data, they’re encoded using the above URL encoding rules. This avoids ambiguity when your program translates the form data. To properly decode this data, you should first split it into *name=value* pairs (eliminating the ampersands), then split each pair into a name and a value, and then apply URL decoding to each portion of the pair.

When a form is submitted, you can often use the order that the form items appear in the form to determine what order in which your CGI program receives the *name=value* pairs. However, you should not depend on this behavior. The various form elements have their own ways to determine what value will be associated with the name they are given. All of the textual input areas use the user’s typed input as the value. Radio buttons use the value of whichever button is enabled. If checkboxes are unchecked, they will use either an empty value string, or their name won’t appear in the encoded form data at all. Hidden form elements can be used to send constant or per-document data to your script without the user’s knowledge or intervention.

The example program, documented in “An Example CGI Program in ANSI C” on page 128, demonstrates decoding of form data.

Information Provided by the Server

As described previously, CGI uses two mechanisms to provide data from the client to your program: the standard input, and environment variables. Let's tackle environment variables first.

Environment Variables

Environment variables are used to pass data about a request to your program. This data is derived from the server software itself, from the network socket connecting the client to the server, and from the URL that was used to access the CGI program.

Accessing Environment Variables

There are a number of different mechanisms programs use to access environment variables, and the mechanism you use is determined by what programming language you use. Environment variables are identified by character strings, and have character string values. The following are some examples of the different mechanisms different programming languages use to access environment variables.

Using C or C++

In C or C++, you can use the getenv library call to access the environment variables.

```
#include <stdlib.h>

char *rhost = getenv("REMOTE_HOST");
```

Using PERL

In PERL, environment variables are accessed through a simple array.

```
$rhost = $ENV{'REMOTE_HOST'};
```

Using the Bourne Shell (/bin/sh)

In the Bourne shell, environment variables are accessed just like normal shell variables.

```
RHOST=$REMOTE_HOST
```

Using the C-Shell

The C shell is similar to the Bourne shell, but it needs the keyword set before any variable assignment.

```
set RHOST = $REMOTE_HOST
```

Variable Formats

The following is a list of each of the environment variables, and their format.

SERVER_SOFTWARE

Gives the name and version of the software which your program is running under.

Format: *name/version*

Example: `Netsite/1.0`

SERVER_NAME

Gives the hostname or IP address of the server machine.

Format: a fully qualified domain name or IP address

Example: `192.82.208.8` or `www.sgi.com`

SERVER_URL

Gives the URL that people should use to access this server. This variable is not supported by revision 1.1 of the CGI interface.

Note: Iis only available using Netsite server software.

Format: *protocol*`://`*hostname*`:`*port*

If the server is running on a protocol's default port, the `:`*port* section won't be present.

Example: `http://www.sgi.com`

GATEWAY_INTERFACE

The revision of the CGI specification supported by the server software.

Format: `CGI/`*n.n*

n.n is the numerical revision.

Example: `CGI/1.1`

SERVER_PROTOCOL

The name and revision of the protocol being used by the client and server.

Format: *name/version*

Example: `HTTP/1.0`

SERVER_PORT

The port number to which this request was sent.

Format: a number between 1 and 65,535

Example: `80`

REQUEST_METHOD

The HTTP protocol defines a number of different methods which are used when accessing URLs on the server. When a person clicks on a hyperlink, the GET method is used.

When a form is submitted, the method used is determined by the METHOD attribute to the FORM tag. There is more information on this in the part of this appendix that discusses processing HTML form data (see “Accepting User Input from URLs and Other Sources” on page 114).

Format: *method*

Examples: GET, HEAD, POST.

CGI programs do not have to deal with the HEAD method directly and can treat it just like the GET method.

PATH_INFO

The extra path information which the server derives from the URL used to access the CGI program.

Format: */dir1/dir2…*

Example: `/html/graphics/doc1.gif`

PATH_TRANSLATED

The Netsite server makes a distinction between pathnames used in URLs, and filesystem pathnames. It is often useful to make your PATH_INFO a virtual path, so that the server provides a physical pathname in this variable. This allows you to avoid giving filesystem pathnames to remote client software.

Format: */dir1/dir2…*

Example: Suppose the document root is set to `/d/netsite/docs`, and the URL `/blat.cgi/doc1.html` is used to access the CGI program. In this case, PATH_INFO is `/doc1.html` and PATH_TRANSLATED is `/d/netsite/docs/doc1.html`.

SCRIPT_NAME

If your program needs to refer the remote client back to itself, or needs to construct anchors in HTML referring to itself, you can use this variable to get a virtual path to your program.

Format: */dir1/dir2/progname*

Examples: `/orders/tickets.cgi`, `/cgi-bin/order-tickets`

QUERY_STRING

When a client accesses an HTML page with a form in it which uses the GET method, or an HTML page that contains the ISINDEX tag and the user executes a search, the information the user provided in the form or search dialog is given to your script in this variable. This information is provided by the server as it was sent by the client, which means it is encoded using the URL encoding rules as described earlier.

Format: varies

Example: From a form, you might get `button1=on&button2=off`, or from a document which contains the `ISINDEX` tag you might get `two+words`.

REMOTE_HOST

This returns the hostname of the remote client software. This is a fully-qualified domain name such as `www.sgi.com` instead of just `www`.

Format: *machine.subdomain.domain*

Example: `core.sgi.com`

If no host name information is available, the script will have to rely on the `REMOTE_ADDR` variable instead.

REMOTE_ADDR

This returns the IP address of the remote host. This information is guaranteed to be present.

Format: *n.n.n.n*

n is a number between 1 and 255.

Example: `192.82.208.8`

AUTH_TYPE

The Netsite server supports HTTP basic access authorization, and in the future may support additional types of authorization. If the script is protected by any type of authorization, this variable will be set to identify the type.

Example: `basic`

REMOTE_USER

If and only if HTTP access authorization has been activated for this script's URL, then this variable will be set to the name of the local HTTP user of the person using the navigation software. This is not a way to determine the user name of any person accessing your program.

Example: `robm`

CONTENT_TYPE

If a form is submitted with the POST method, then this is the type of data being sent by the client. Note that while clients currently only send `application/x-www-form-urlencoded`, this variable can contain any MIME type. In the future, systems may use this method to transfer data back and forth.

Format: *type/subtype*

CONTENT_LENGTH

This is the number of bytes being sent by the client. Note that while some HTTP servers do not send an EOF at the end of the data, *Netsite does send an EOF*, alleviating the need to count the number of bytes your script has read from the input stream.

Secure Server Variable Formats

The Netsite Commerce Server defines the following additional variables to describe the server and client's security status.

HTTPS

On or off, depending on whether security is active on this server.

HTTPS_KEYSIZE

When security is on, this is the number of bits in the session key used to encrypt the session.

HTTPS_SERVER_ISSUER

The issuer of this server's key pair.

HTTPS_SERVER_SUBJECT

The subject of this server's key pair.

HTTP Headers

In addition to the environment variables, if the client sends any additional HTTP headers along with its request, then these headers are also placed into the environment. The only exception is the Authorization header. The HTTP header is prefixed with `HTTP_`, and have all of its letters changed to upper case and all dash (-) characters changed into underscore (_) characters. Examples of these are shown here.

HTTP_ACCEPT

This header enumerates the types of data the client can accept. For most client software, this protocol feature has become a bit convoluted and this information isn't always very useful.

Format: *type/subtype[, type/subtype]…*

Example: `image/gif`, `image/jpeg`, `*/*`

HTTP_USER_AGENT

This identifies the software being used to access your program.

Format: varies

Example: `Mozilla/0.9b (Windows)`

HTTP_IF_MODIFIED_SINCE

The client requests that the program's response be sent only if the data has been modified since the given date. This date is set according to GMT standard time.

Format: *Weekday, dd-mon-yy hh:mm:ss GMT*

Weekday specifies the full name of the day, such as Thursday or Friday. *dd* specifies the number of the day of the month. *mon* specifies the 3-letter abbreviation of the month. *yy* specifies the current year within the century. *hh:mm:ss* gives the current time, in 24-hour format.

HTTP_FROM

This is the e-mail address of the remote user. If their client software supports this header the client software can send it. This may be user selectable depending on the kind of client software being used.

Format: *user@machine.subdomain.domain*

Example: `robm@sgi.com`

The Standard Input

As discussed previously in the section about translating form data, HTML forms which use the POST method send their encoded information using the standard input. In this case, the standard input will have your form data. If you do not need to worry about your program running with other server

software, you can simply read this data until receiving an end of file. Although the only currently used content-type is `application/x-www-form-urlencoded` you may use the standard input with a custom navigator or application to send other types of data to your programs via the standard input.

Program Output

As shown at the beginning of this appendix, your script's output goes through the server-spawned CGI process. The server usually does things to make your CGI program's life easier, by speaking the client's protocol, so you don't have to worry about the output. This makes your programs simpler and guarantees that they can take advantage of newer revisions of the protocol with little or no changes.

However, if you know enough about the HTTP protocol to code the protocol and send it directly back to the client, and you absolutely need to do so, you can use the non-parsed header feature. As of CGI/1.1, the only reason you would need this feature is if your program outputs an excessively lengthy amount of data, and you want to sidestep the server's buffering of your output. To activate the feature, make your script's name start with the characters `nph-` and the server will make the standard output a direct copy of the socket to the client. At this point your program is responsible for any protocol-related response headers or messages.

Regardless of the fact that you can do this, in most cases your programs will want to avoid this feature. CGI programs must print a valid CGI header on the standard output for the server to accept their response and send it on to the client. In the Netsite server, the standard output and the standard error file streams are directed to the same place: back into the server.

This means that errors your program generates or system utilities your program call can interfere with your header. Similarly, if your program is abnormally terminated (through a bug or some other disaster), the server will send a server error to the client, and describe the error in the server's error log. Because of this, you will want to print your header as early as possible in your program.

CGI Generic Headers

A CGI header consists of several text lines, of the form:

```
name: value
```

The end of the header is signaled by a single blank line. After the blank line, the server stops parsing your program's header and sends the rest of your data untouched back to the client. This means that your program can output any type of data it needs to, including HTML, GIFs, or JPEGs.

Each name: value pair is an HTTP protocol header. You can output any header you wish, and the server will send it back to the client. Some of the commonly used HTTP headers are described here.

Content-length

The length of your data in bytes, not including the header.

Content-type

The type of data your program is returning. This is a valid MIME type in the format *type/subtype*. This header should always be sent from any CGI program.

Examples: `text/html`, `text/plain`, `image/gif`, `image/jpeg`, `audio/basic`

Expires

Gives the date on which this file should be considered outdated by the client. The date format is the same as the format for `if-modified-since`.

Example: `Saturday, 12-Nov-94 14:05:51 GMT`

Content-encoding

The data is the given content-type, but it is compressed. Current values that can be used are `x-gzip` for GNU zip compression, and `x-compress` for standard UNIX compression.

When you output any of the above headers, the server will not alter their values or their output.

CGI Specific Headers

On the other hand, the following headers are special to CGI and cause the server to take an action on your behalf.

Location

Location gives the location of a new file to have the server or client retrieve. This header must be in one of two forms.

If the value is a virtual path, such as `/misc/file.html`, then the server re-starts the request as if the client had originally requested:

```
http://yourserver/misc/file.html
```

Note that the client won't be informed about this, however, and any relative links will be resolved from the directory of your CGI program, not of the document that is actually being returned.

If the value is a full URL, such as:

```
http://yourserver/misc/file.html
```

then the server redirects the navigator to the new URL. The navigator then acts as if it had originally requested that URL, and all relative links are resolved from the directory specified in that URL. You do not necessarily have to redirect to an HTTP URL, you can redirect to a gopher, news, FTP, or any other valid URL.

Status

Every HTTP request is returned with a status code which indicates to the client whether the request succeeded or not. If the request was unsuccessful, several error codes are provided to tell the client what happened. If the request was successful, there are also status codes to indicate a successful request and request further action from the client. If no Status line is

provided, the default is `200 OK` unless a Location header with a full URL is present. If the location is present, the default is `302 Found`.

The status line has the form *nnn reason*, where *nnn* is the 3-digit code for the request, and *reason* is a short string describing the error. The following codes and reasons are currently recognized by Netscape Navigator.

- `200 OK`

  The request finished normally.

- `204 No response`

  The request was understood and processed, but there is no new document to be loaded by the client.

- `302 Found`

  The client should look for data at a new URL, given by a Location header.

- `304 Use local copy`

  The client sent a request with an `if-modified-since` header, and the requested data hasn't been modified since the given date.

- `400 Bad request`

  The request had illegal or unintelligible HTTP inside.

- `401 Unauthorized`

  If access authorization is enabled, the request could not be fulfilled because the user did not provide the proper authorization to access the area. With current authorization schemes, a `WWW-Authenticate` header must be provided to give the client instructions on how to complete the request with the proper authorization.

- `403 Forbidden`

  The client is not allowed to access what it requested.

- `404 Not found`

  The client asked for something the server couldn't find.

- `500 Server error`

  This is a catch-all error code which indicates that something has gone wrong in the server or the CGI program, and that problem stopped the request from being completed.

- `501 Not implemented`

  The client asked the server to perform an action which the server knows about, but can't do.

Sample Program Output

The following CGI program output would send an HTML document back to the client:

```
Content-type: text/html

<title>My little document</title> This is my own little
document. Do you like it?
```

The following output would instruct the client to retrieve a different URL. The small HTML fragment at the bottom is customary, and allows navigation software which doesn't support redirection properly to retrieve the given URL.

```
Location: http://foobar.org/boards/fromitz

This document can be accessed at the following <a
href=http://foobar.org/boards/fromitz>location</a>.
```

An Example CGI Program in ANSI C

Here is an example CGI program which handles the output of an HTML form. Here is the text of the form, presented without the markup is:

```
<TITLE>Guest book</TITLE>

<H1>Guest book</H1>

Welcome to our guest book! Choose one of the following:<p>

<FORM ACTION=gb.cgi METHOD=POST>

<INPUT TYPE=radio NAME=action VALUE=log> Log your name to
the guest book: <INPUT TYPE=text NAME=email SIZE=40><p>

<INPUT TYPE=radio NAME=action VALUE=read> Read the guest
book<p>

<INPUT TYPE=submit VALUE="Let's do it!">
```

```
</FORM>
```

The user would first access the above HTML form, as */guestbook/gb.html*. Once the user selects an action, and optionally types in their name for the guest book, they click on the submit button. Note that the name of the radio button is *action*, and can have the values *log* or *read*. Also note that there is a text area called *email*.

Figure C-5 Sample Guest-Book / gb.html as It Would Appear in the Client Window

Here is the program that is called when the user submits this form. In this example, the server administrator has activated CGI as a file type, and so the compiled program which goes with this source code is called *gb.cgi*.

For brevity, the example does not include the code necessary to actually update or maintain a guest book. It shows only the code necessary to get information to and from the server and client through CGI.

Example C-1 Sample CGI Program (Part 1)

```
/*
 * gb.c: A short CGI example program in C.
 *
 * This takes the output of a form and does something with it. If the user
 * entered their name, then it enters it into an imaginary guest book.
```

```
 *
 * If they decided to view the guest book instead, they are redirected to
 * a new URL. The function to update the guest book is not actually
 * provided here, in order to keep the focus on CGI related things.
 *
 *
 * Rob McCool
 */
```

The comments to the right of these header file includes indicate which function prototypes are used from that header file.

Example C-2 Sample CGI Program (Part 2)

```
#include <stdio.h> /* stdin, printf, fread */
#include <stdlib.h> /* malloc, getenv */
#include <ctype.h> /* isalpha */
#include <string.h> /* strchr, strcmp */
```

Start out with some defines and prototypes. Code execution will begin in the main function below.

Example C-3 Sample CGI Program (Part 3)

```
/* The biggest set of form data we'll accept. This is a small form */
#define MAX_CONTENT_LEN 2048


/* Function defined below which prints an error to the client and exits */
void err(char *s);

/*
 * Function defined below which takes a pointer to some URL-encoded
 * data and allocates a new string, copies and decodes the data into
 * that string, and returns the new string.
 */

char *url_decode(char *encoded);


/* This function logs the given e-mail address to the guest book */

void log_address(char *email)
{
    /* This code is not provided... it should be straightforward */
```

```
}

int main(int argc, char *argv[])
{
```

The above functions and defines will be used in the main subroutine. This is where the program starts. Note that the command line parameters to main are defined, but not used. It begins by defining and initializing its variables. The variables that are not initialized here are initialized below as they are needed.

Example C-4 Sample CGI Program (Part 4)

```
    /*
     * Get the method used to access the script from the environment. This
     * form uses the POST method, and using any other results in an error.
     */
    char *method = getenv("REQUEST_METHOD");
    /* The length of the form results the browser sends us (set below) */
    char *clstr;
    int clen;
    /* We hold the form contents in this buffer (+1 for the null) */
    char content[MAX_CONTENT_LEN + 1];
    /* Return code from system or library calls */
    int ret;
    /* Pointers used to parse the form data */
    char *name, *value;

    /* A pointer for the remote person's e-mail address */
    char *email = NULL;
    /*
     * Which action the user selected. -1 is invalid and used to indicate
     * that no action was selected.
     */
    int action = -1;
```

The code begins here. The first thing this program does is check the data the server provides in the environment to see if it is correct. If it is not, the request cannot proceed and a message is sent to the navigation software.

Example C-5 Sample CGI Program (Part 5)

```
    /* First, see if they got the method right. */
    if(strcmp(method, "POST") != 0)
        err("you must submit a <a href=gb.html>form</a> to access this URL.");
```

```
/* Make sure we got form data and check its length */
clstr = getenv("CONTENT_LENGTH");
if(!clstr)
    err("your browser didn't send any content. Is it not POST capable?");
/* Change that string into a number */
clen = atoi(clstr);

/* Make sure it's really form data */
if(strcmp(getenv("CONTENT_TYPE"), "application/x-www-form-urlencoded") != 0)
    err("your browser sent the wrong content type.");

/*
 * Check for negative or outrageously large content lengths
 * An upper limit is set to make sure they don't steal extra system
 * resources for no good reason
 */
if((clen < 0) || (clen >= MAX_CONTENT_LEN))
    err("your browser created too much data from that tiny form.");
```

The variables appear to be set correctly. The program can now read the form data into a string variable.

Example C-6 Sample CGI Program (Part 6)

```
/* Read in the data in one shot */
ret = fread(content, 1, clen, stdin);
/* Return of < 1 means either EOF or error */
if(ret < 1)
    err("an I/O error occurred before your form data could be read.");
/* Terminate it with a null char */
content[ret] = '\0';
```

Now that the program has the string of data the client sent it, the program parses that string. This involves splitting the string into name=value pairs, then splitting those pairs into name and value strings, and then URL-decoding those strings.

Example C-7 Sample CGI Program (Part 7)

```
/*
 * Here's where the fun starts. Most of the time, you will want to create
 * a generic function to decode form data, and then use that routine in
 * your scripts.
 *
```

```
 * Form data looks like this:
 *
 * name1=value1&name2=value2 ...
 *
 * The nameN and valueN strings are URL-encoded, which means that many
 * special characters such as spaces, semicolons, forward or back slashes,
 * question marks, percent signs, newlines, plus signs, and colons will be
 * changed from one character into three, of the form %xx, where x is a
 * hexadecimal digit. These two digits are changed into a number from 0-255
 * which is interpreted as a character.
 *
 * Further, many browsers turn spaces into plus signs.
 */

/* Start our name pointer at the beginning of the data */
name = content;
/* name will be set to NULL by some of the code below when we're done */
while(name && (*name != '\0')) {
    /* We first find somewhere to put the value pointer */
    value = strchr(name, '=');
    /* However, we must check to make sure we got the right data */
    if(value == NULL)
        err("the submitted form data was corrupt.");

    /* Otherwise, mark the end of the name string. */
    *value++ = '\0';

    /*
     * We now have the name string by itself. See what it is.
     * Note that we do not URL-decode the name string, because our form
     * is set up such that the names don't use any bad characters.
     */
    if(strcmp(name, "action") == 0) {
        /* First, find the next name string. NULL indicates the last one */
        name = strchr(value, '&');
        if(name != NULL)
            *name++ = '\0';

        /*
         * Action can have two values: "log" and "read". Again, we
         * avoid URL-decoding the value because we don't use any
         * special chars in our form.
         */
        if(strcmp(value, "log") == 0)
            action = 1;
```

```
        else if(strcmp(value, “read”) == 0)
            action = 2;
        else
            err(“your form results had an invalid action.”);
    }
    else if(strcmp(name, “email”) == 0) {
        /* Find the next name string. NULL indicates the last one */
        name = strchr(value, ‘&’);
        if(name != NULL)
            *name++ = ‘\0’;

        /*
         * The email value will be an e-mail address, which can have
         * weird % signs and ! marks in them. We have to URL decode
         * this string. url_decode is defined below.
         */
        email = url_decode(value);
    }
}
```

Now that the program has the data in the email and action variables, it can verify that the form was filled out in its entirety.

Example C-8 Sample CGI Program (Part 8)

```
/* Check to make sure they filled out the necessary data. */
if(action == -1)
    err(“you did not pick a button.”);
if((action == 1) && ((email == NULL) || (*email == ‘\0’)))
    err(“you did not fill out your e-mail address”);
```

Finally, the program looks at the action the user selected and does what they asked it to do. Note that there is an example of returning a new document (a success page in HTML) and an example of redirecting the client to a new URL (the one that has the guest book for them to view).

Example C-9 Sample CGI Program (Part 9)

```
/* Now that we have the data, do something with it. */
if(action == 1) {
    /* Log their e-mail address */
    log_address(email);

    /* Now generate a success page */
    printf(“Content-type: text/html\n\n”);
```

```
        printf("<title>Congratulations</title><h1>Congratulations</h1>\n");
        printf("Your name has been added to our guest book, %s!\n", email);
    }
    else {
        /* View the guest book */

        /*
         * Instead of printing a new HTML page, this function sends them to
         * a new URL. This URL is hard-coded to a certain location, but uses
         * CGI variables to get the server's hostname and port.
         */

        printf("Location: http://%s:%s/guestbook/guests.html\n\n",
        getenv("SERVER_NAME"), getenv("SERVER_PORT"));
    }
    return 0;
}
```

Below are the supporting functions that the main program calls to perform its basic tasks.

Example C-10 Sample CGI Program (Part 10)

```
/* --------------------------- Function: err ---------------------------- */


/* Returns an error to the client */
void err(char *s)
{
    /* Print the CGI header telling the client that this is HTML */
    printf("Content-type: text/html\n\n");

    /* This generates HTML for the client, telling them what went wrong. */
    printf("<title>Guestbook error</title><h1>Guestbook error</h1>\n");
    printf("Your form results could not be processed because %s.\n");

    /* Now, just exit and let them try again */
    exit(0);
}
```

The following functions look a lot scarier than they are. They perform URL decoding on the string.

Example C-11 Sample CGI Program (Part 11)

```
/* ------------------------ Function: url_decode ------------------------ */


/* First, a function that verifies that a 2-char string is a hex digit */
int is_hex(char hex)
{
    /* Make sure it's lower case */
    if(isalpha(hex))
        hex = toupper(hex);

    /* This just checks the character to see if it's in the two ranges */
    if(((hex < 'A') && (hex > 'F')) && ((hex < '0') && (hex > '9')))
        return 0;
    else
        return 1;
}

char *url_decode(char *encoded)
{
    /* Allocate space for the new string. We won't need more space than we
       already have after decoding */
    char *new = (char *) malloc((strlen(encoded) + 1) * sizeof(char));
    /* Index register for string copy */
    char *enc, *dec;
    /* Digit is used to translate hex into character */
    char digit;

    if(new == NULL)
        err("the program ran out of memory.");

    /* We go through the string, looking for + signs or percents */
    for(enc = encoded, dec = new; *enc; enc++, dec++) {
        if(*enc != '%') {
            /* Plus goes to space, but most chars are untouched */
            if(*enc == '+')
                *dec = ' ';
            else
                *dec = *enc;
        }
        else {
            /* Another tricky part. First, make sure we got what we want. */
            if((!is_hex(enc[1])) || (!is_hex(enc[2])))
                err("invalid escape sequence");
```

```
            /* Now, advance over the % sign to the first digit, and decode. */
            /* The 0xdf is to turn the character into upper case */
            ++enc;
            if(*enc >= 'A')
                digit = 16 * (((*enc & 0xdf) - 'A') + 10);
            else
                digit = 16 * (*enc - '0');

            /* Now, advance to the second digit and decode. */
            ++enc;
            if(*enc >= 'A')
                digit += ((*enc & 0xdf) - 'A') + 10;
            else
                digit += *enc - '0';

            /* Finally, transfer the digit to the new string. */
            *dec = digit;
        }
    }
    *dec = '\0';

    return new;
}
```

Tips for CGI Program Development

When developing CGI programs, the first thing you will notice is that you can't effectively use a debugger to find out what's wrong with your programs. A primitive though effective solution to this problem is to use print statements in your program which print the contents of a variable to the client. Note that you'll need to print a small CGI header before you print the value of a variable if you haven't already printed a header in your program yet.

If you are using C to write your program, and you have the *dbx* or *cvd* debugger installed on your system, then it is possible to attach to your program before it begins doing its work. That is, place a call to *sleep* at the beginning of your program, and make that *sleep* long enough for you to search the process list for your program and attach to that process with your debugger.

Finally, although the CGI specification does not explicitly require it, the Netsite server changes its current directory to the directory in which the CGI program resides. This means that if your program dumps core, the core file can be found in the directory in which the program is executing (if the user the server is running as can write to that directory).

Appendix D

Technical Information

This appendix is intended to provide technical information for editing the configuration files directly with a text editor or are simply interested in how the files work. You'll find descriptions of:

- Manual configuration
- Example configuration files
- The technical configuration file
- The Init directive
- The object configuration file

Manual Configuration

On occasion, you may need to configure your server by hand. One such case is if you forget your administrative password. Another is if you accidentally lock your hosts out of the administrative forms.

To find your way out of these binds, familiarize yourself with the various Netsite server configuration files. These files are kept in the directory *admin/config* in your server root. The following files are in that directory:

- *magnus.conf*: the server's main technical configuration file. This file controls aspects of the server operation not related to documents, such as hostname and port. This file is described in detail in "The Technical Configuration File" on page 140.
- *obj.conf*: the server's object configuration file, which controls how the server finds your documents. This file is described in detail in "The Object Configuration File" on page 156.
- *mime.types*: the file the server uses to convert file name extensions such as *.gif* into a MIME type like *image/gif*.

- *admpw*: administrative password. Format is `user:password`. The password is DES-encrypted in the same fashion it would be in */etc/passwd*. If you ever 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.

To see how it all fits together, take a look at the example configuration files.

Example Configuration Files

After you install your server with the Netsite QuickStart installation forms, you will find that the server has already written the *magnus.conf* and *obj.conf* configuration files. Understanding these files can help you understand how to set up (and change) the basic functionality of the server. Figures D-1 through D-4 show the example files.

The Technical Configuration File

The technical configuration file, called *magnus.conf*, controls those aspects of server operation which are not related to specific documents or directories of documents. All of the items in this configuration file are global and apply to the entire server, as opposed to affecting only one directory or set of directories.

Each line has the format:

```
Directive value
```

Comment lines must begin with a # character, with no leading white space. Directive lines may contain a mix of white space at the beginning of the line and between the directive and value, but trailing white space after the value may confuse the server. Long lines (which should only occur with the Init directive) can be continued with a \ character just before the line feed.

Directive identifies which aspect of server operation you are changing. This string is case insensitive.

value is a specific value you are giving the directive. Its format depends on the directive. This string is usually case sensitive.

Example D-1 Sample of magnus.conf

```
# In this file, any line beginning with a # is a comment.
# Note that ServerRoot is not a directive to the server itself, but the
# administrative forms put this line in there to keep themselves in sync.
#ServerRoot /var/mc-httpd
# This server is running on the default HTTP port.
Port 80
# Declares that obj.conf should be loaded as our object configuration file
LoadObjects obj.conf
# Within obj.conf, the object named default will be the server's root
# object. Global configuration for the server will be derived from
# this object.
RootObject default
# Sends errors to a file
ErrorLog /var/mc-httpd/logs/errors
# Logs the pid where the admin forms expect them to be
PidLog /var/mc-httpd/logs/pid
# After startup, run as the user named netsite
User netsite
# No matter what the hostname, make sure URLs reference www.sgi.com
ServerName www.sgi.com
# We only want 32 server processes running at a time.
MaxProcs 32
# Loads mime.types as the filename extension to MIME type database
Init fn=load-types mime-types=mime.types
# Opens an access log in the logs subdirectory. This log is internally
# named global, which will be important below.
Init fn=init-clf global=/var/mc-httpd/logs/access
```

The Directives

This section defines the directives, and describes their characteristics including: directive name and description, caveats (or things to watch out for), format for value string, default value if the directive is omitted, how many instances of the directive should be in the file and examples. The directives covered here include:

- ServerName
- Port
- User
- MaxProcs
- ErrorLog
- PidLog
- LoadObjects
- RootObject
- Chroot

ServerName

Name and Description

ServerName tells the server what to put in the hostname section of any URLs it sends back to the client.

Caveats

The setting of this directive does not affect the URLs you write to your server, but only affects the URLs the server automatically generates. If you want to be able to take a machine named:

```
machine.[yourdomain].[dom]
```

and access it as

```
www.[yourdomain].[dom]
```

in your URLs, this flag is only part of the work you need to do. You also need to have your system or network administrator set up a DNS CNAME or alias which makes

```
www.[yourdomain].[dom]
```

a hostname pointer to

```
machine.[yourdomain].[dom]
```

Syntax

ServerName *host*

host is a fully qualified domain name such as

```
yourhostname.[yourdomain].[dom]
```

Default

If no **ServerName** is given, the server will attempt to derive a hostname through various system calls. If this fails to return a fully qualified domain name (for example. it gets *yourhostname* instead of *yourhostname.[yourdomain].[dom]*), the server will not start up and will ask you to set this value manually.

How many?

There should be one or zero **ServerName** directives in the configuration file.

Examples

```
ServerName www.sgi.com
ServerName www.company.com
ServerName www.agency.gov
```

Port

Name and Description

Port controls which TCP port the server listens to.

Caveats

If you choose a port number less than 1024, the server must be started as root or superuser.

The port you choose affects URLs you write to your server. If you choose a port number other than 80, you must include that port number in your URLs like so:

```
http://www.foobar.org:8080/
```

If you choose port 80, your URLs may be written like so:

```
http://www.foobar.org/
```

Syntax

Port *number*

number is a whole number between 0 and 65,535

Default

If no **Port** is given, the server assumes **Port 80**

How many?

There should be one or zero Port directives in the configuration file.

Examples

```
Port 8080
Port 8000
Port 1000
```

User

Name and Description

User controls which of your IRIX system's users[1] the daemon will run as. If it is started by the superuser, the server will bind to the Port you give it, and then switch its effective user ID to the user you specify here. The user you select should have no special privileges because privileges get passed on to server processes and are inherited by other objects. Were the server to enter a fault state, you would want it to have limited privileges.

Caveats

This directive is ignored if the server is not started as the superuser.

The user must exist in the system's passwd database, and have a valid user ID. You can choose the user *nobody* if you wish, but this might not work on all systems. Some systems define the user *nobody* such that its user ID is invalid. A *uid* less than zero is invalid and generates an error during startup. Check the */etc/passwd* file to see if the uid for *nobody* exists and that it is greater than 0. This same principle holds true for any Unix user you assign as the server user.

Syntax

User *login*

login is the 8 character or less login name of one of your system's users.

[1] Note that this is a Unix user, *not* an HTTP user.

Default

If no **User** is given, the server will run as whichever user it is started as. If it is started as root, and no **User** directive is present, the server will print a warning that it is running as the superuser.

How many?

There should be one or zero **User** directives in the configuration file.

Examples

```
User rob
User nobody
User netsite
```

MaxProcs

Name and Description

MaxProcs sets the number of processes the server has active at any given time. Netsite always starts up the number of **MaxProcs** processes indicated with the directive but it won't start any additional processes. It replaces any children which exit, up to the number indicated with **MaxProcs**.

Caveats

Choosing a number which is too low for this directive can result in a low system load for your server machine, but may expose clients to unnecessary delays in service.

Choosing a number which is too high may not cause serious problems, but can tie up more resources than necessary on your server machine. This might be an issue on a machine which is not a dedicated server.

Syntax

MaxProcs *number*

number is a number between 1 and the size of your system's process table

Default

If no **MaxProcs** directive is given, the server assumes **MaxProcs 50**

How many?

There should be one or zero **MaxProcs** directives in the configuration file.

Examples

```
MaxProcs 16
MaxProcs 8
MaxProcs 64
```

ErrorLog

Name and Description

ErrorLog tells the server where to log its errors. The syslog facility can be used if desired.

Caveats

If errors are to be reported to a file (instead of syslog), then the file and directory in which the log is kept must be writable by whatever User the server is running as.

Syntax

ErrorLog *log*

log is either a full pathname to a log file, or the keyword *SYSLOG* which indicates that errors should be reported through the syslog facility.

Default

There is no default error log.

How many?

There should be one **ErrorLog** directive in the configuration file.

Examples

```
ErrorLog /var/mc-httpd/logs/errors
ErrorLog SYSLOG
```

PidLog

Name and Description

PidLog specifies a file in which to record the process ID of the base server process. To shut down your server, you should *kill* the base server process with a **-TERM** signal. To tell your server to re-read its configuration files and re-open its log files, use *kill* with the **-HUP** signal.

Caveats

If the file you give here is not writable by the user the server is started as, the server doesn't complain or log its pid anywhere.

Some of the server support programs assume that the pid log will be kept in the server root, under *logs/pid*.

Syntax

PidLog *file*

file is the full pathname of the file to log the server process ID in.

Default

There is no default **PidLog**.

How many?

There should be one or zero **PidLog** directives in the configuration file.

Examples

```
PidLog /var/mc-httpd/logs/pid
```

```
PidLog /tmp/mc-httpd.pid
```

LoadObjects

Name and Description

LoadObjects parses an object configuration file, which (among other things) tells the server where to find documents.

Caveats

The administration forms assume that there is only one object configuration file and that it will be in the server root, under:

admin/config/obj.conf

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 was given, the server looks in the current directory.

Default

There is no default **LoadObjects** directive.

How many?

There should be one or more **LoadObjects** directives in the configuration file.

Examples

```
LoadObjects obj.conf
```

```
LoadObjects /var/mc-httpd/admin/config/local-objs.conf
```

RootObject

Name and Description

RootObject 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, and any server behavior which is configured in the default object affects the entire server.

Caveats

If you name an object which 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 **RootObject**.

How many?

There should be one **RootObject** directive in the configuration file.

Examples

```
RootObject default
RootObject server1
```

Chroot

Name and Description

Chroot allows the administrator to place the server into a jail, where it is only allowed to access files from a given directory. The idea is that if the server's security is ever compromised, and an intruder managed to obtain shell access on the server machine, that intruder would only be able to affect a very limited set of documents on the server machine.

Caveats

The server must be started as the superuser to use the **Chroot** directive.

If the **Chroot** directive is used, CGI C programs must be linked statically. If any system binaries are used (such as *perl* or */bin/sh*) they must be copied to the chroot directory.

A chroot server cannot be restarted with the **-HUP** signal.

You won't be able to use the Netsite Server Manager forms on a chroot server.

The user public information directory feature will not be available unless a copy of */etc/passwd* is kept in the chroot directory, and all of the users' home directories are exactly mirrored within the chroot directory.

Logs and server configuration files should be kept outside the chroot directory.

Syntax

Chroot *directory*

directory is the full pathname of the directory to use as the server's root directory.

Default

There is no default **Chroot**.

How many?

There should be one or zero **Chroot** directives in the configuration file.

Examples

```
Chroot /d/mc-httpd
Chroot /www
```

The Init Directive

Init is a special directive which initializes certain server subsystems such as access logging and user public directories. Generally, the purpose of these functions is to load data for specific subsystems once on server startup, and to keep that data internally until the server is shut down. Zero or more Init directives may appear in the technical configuration file.

Init directives have the form:

```
Init fn=function-name [parm1=value1] ... [parmN=valueN]
```

function-name identifies which server initialization function should be called. Subsystem initializers should not be called more than once.

parm=value pairs are each values to give function-specific parameters. Depending on the function, you will need zero or more of these parameters. The order of parameters on the line does not matter. In fact, `fn=function-name` may appear anywhere among the parameters.

Description of Init functions

This section defines the Init functions, and gives their characteristics, including: name and description, importance, caveats, required parameters, optional parameters and examples. The Init functions covered here include:

- File Name Extension to MIME Types
- Logging
- User Public Info Dirs
- Directory Indexing

File Name Extension to MIME Types

Name and Description

The function **load-types** scans a file which tells it how to map filename extensions to MIME types. These types are essential for network navigators like Netscape Navigator to be able to tell the difference between an HTML file and a GIF file, for instance.

Importance

Calling this function is crucial unless your server only serves one type of file.

Caveats

None.

Required Parameters

mime-types specifies either the full filesystem path of the global MIME types file, or a file name relative to the server configuration directory. This file is supplied with the server and called *mime.types*.

Optional Parameters

local-types is in the same format as the global MIME types file, but can be used to maintain types which may only be applicable to your particular site.

Examples

```
Init fn=load-types mime-types=mime.types
Init fn=load-types mime-types=/foo/mime.types local-types=local.types
```

Logging

Name and Description

The function **init-clf** initializes the Common Log subsystem. It opens the log files whose names are given as parameters, and these log files stay open until the server is shut down, or until the base server process is sent the **-HUP** signal, at which time the logs will be closed and re-opened.

Importance

Initializing this function is required if you are using the common log feature of the server.

Caveats

If you move, remove, or otherwise disturb the log file without shutting down or restarting the server, client accesses may not be recorded.

Parameters

At least one log file should be given. The *parm* part of the *parm=value* pair should be a unique name for the log file. You will use this name later on, as a parameter to the **common-log** function.

Examples

```
Init fn=init-clf global=/var/mc-httpd/logs/access
Init fn=init-clf global=/tmp/httpd-access cgi=/tmp/script-log
```

User Public Information Directories

Name and Description

The function **init-uhome** loads information about the system's users' home directories into internal hash tables. This increases the shared memory size

a bit, while saving CPU cycles for servers which have a lot of traffic to and from home directories.

Importance

Calling this function is entirely optional.

Caveats

None.

Optional Parameters

pwfile specifies the full filesystem path of a file to use other than */etc/passwd*. If no **pwfile** is given, the system default (use NIS or look in */etc/passwd*) is used.

Examples

```
Init fn=init-uhome
Init fn=init-uhome pwfile=/etc/passwd-http
```

Directory Indexing

Name and Description

The function **cindex-init** sets certain global characteristics of the fancy indexing subsystem.

Importance

Calling this function is completely optional.

Caveats

You must be using fancy indexing for this function to have any meaning.

Optional Parameters

opts specifies certain options to be activated for indexing. This value is a string of letters, one for each option to activate. **i** makes all icons links. **s**

makes the server scan HTML documents in the directory it is indexing in order to place their titles in the description field.

widths specifies the width for each column in the indexing printout. A width of zero disables that column. The **widths** string should be a comma-separated list of numbers, specifying the column widths corresponding to name, last modified date, size and description. The length of size and last modified is fixed (that is,giving any non-zero width activates the column).

ignore gives a wildcard pattern of file names to ignore while indexing. File names starting with a dot are automatically ignored.

icon-uri gives the prefix the server should use for icons. By default, this is */mc-icons/*. The server looks in this directory for the GIF files to use in fancy indexing.

Defaults

If you don't call this function, the internal data is initialized as if you entered:

```
Init fn=cindex-init widths=22,1,1,33
```

Examples

```
Init fn=cindex-init widths=50,1,1,0
Init fn=cindex-init widths=50,1,1,30 opts=s
Init fn=cindex-init widths=22,0,0,50 opts=is
```

The Object Configuration File

This section is intended to provide a basic overview of how the server deals with your documents, and to provide you with the knowledge you need to understand the object configuration file, usually called *obj.conf*, a sample of which is shown on the following pages.

Definition of Objects

The first thing you should be wondering is just what an object is, since the rest of the documentation does not use this terminology. Internally, the server views all of its contents as a set of objects. Any number of directories, CGI programs, documents, imagemap files, and so on. can be grouped together to form an object for the server. This grouping can then be used to control the behavior of the specified part of your server. You can create any number of objects. Object configuration lets you exercise fine grain control over every part of your server.

Example D-2 Sample of obj.conf

```
# The first object in this file is the object named default. This is
# the server's root object. Any server behavior defined in this object
# which is not overridden by another object affects the entire server.

<Object name=default>

# The first name translation here causes URLs beginning with
# /admin/bin to come from the directory /var/mc-httpd/admin/bin. Note
# the name=cgi reference to the named object cgi. This means that the
# /var/mc-httpd/admin/bin directory will get configuration information
# from the cgi object, and so every file in admin/bin will be treated
# as a CGI object.

NameTransfn=pfx2dir from=/admin/bin dir="/var/mc-httpd/admin/bin"
name=cgi

# Another two name translations, but these do not link to the cgi object
NameTrans fn=pfx2dir from=/admin dir="/var/mc-httpd/admin/html"
NameTrans fn=pfx2dir from=/mc-icons dir="/var/mc-httpd/mc-icons"

# Any request that doesn't start with the above three prefixes will
# come from this directory.
NameTrans fn=document-root root="/usr/netsite-docs"

# The first thing we do is look for nasties like ../ and // in the
# path. Any request that has them is returned a not found error. This
# line is crucial to maintain your system's security.
PathCheck fn=unix-uri-clean

# This looks for any path info attached to the path. If path info is
# found, but the document type doesn't need it (i.e. it's not a CGI
```

```
# program or parsed HTML) then the request will be flagged as not found.

PathCheck fn=find-pathinfo

# If someone references a directory anywhere on the server, this
# directive will cause the server to look in that directory for
# either index.html or home.html. If it finds one, it will use that
# file to send the user an index of that directory.

PathCheck fn=find-index index-names="index.html,home.html"

# This simply tells the server to use the MIME types loaded in
# magnus.conf to determine each file's type.

ObjectType fn=type-by-extension

# If the above directive fails to find the type, force the type to be
# text/plain by default.

ObjectType fn=force-type type=text/plain

# The following three lines cause .map files to be handled as
# imagemaps, directories with no index files to be indexed using the
# common format, and any other non-internal file format to be sent as
# plain files.

Service method=(GET|HEAD) type=magnus-internal/imagemap fn=imagemap
Service method=(GET|HEAD) type=magnus-internal/directory fn=index-common
Service method=(GET|HEAD) type=*~magnus-internal/* fn=send-file

# All accesses to the server will be recorded in the log named global.
AddLog fn=common-log
</Object>

# The following named object is used for implementing CGI directories.
# Note that any file from these directories is typed as a CGI program.

<Object name=cgi>
ObjectType fn=force-type type=magnus-internal/cgi
Service fn=send-cgi
</Object>

# The following is an example of an object specified by a path
# pattern. It's used to protect the administration forms. The
# configuration directives here apply only to the directory
```

```
# /var/mc-httpd/admin.

<Object ppath="/var/mc-httpd/admin/*">

# This directive assigns a user database and HTTP authorization type for
# this directory.
AuthTransfn=basic-ncsa auth-type=basic
userfile="/var/mc-httpd/admin/config/admpw"

# The following cluster is used to hide this directory from anyone
# except *.sgi.com or 127.0.0.1. The *~ at the beginning of each
# pattern means "every pattern except".

<Client dns="*~*.sgi.com" ip="*~127.0.0.1">
PathCheck fn=deny-existence
</Client>

# This is the part that actually requires the authorization.

PathCheck fn=require-auth realm="Server Administration" auth-type=basic
</Object>
```

The server allows you to specify which of your files or directories are part of each object in two ways:

- By an arbitrary name. This is also called configuration-by-template. In order to easily apply the same configuration parameters to an arbitrary number of directories, you can assign a name to the configuration you want these directories to have. You then use the name translation functions to tell the server which directories should use this configuration.

  A good example of this is users' public information directories. If your users have their home directories scattered all over the system, with other servers you would need to list each directory explicitly in the configuration file. You would also have to update your configuration file if a new directory of users was ever added, a process which can be error prone. By contrast, if you assign a named object to be the configuration for your users' public directories, then any user's home directory automatically uses the template configuration.

  Another example is *cgi-bin* directories. In these directories, all files are treated as programs and are executed rather than sent. If you assign a named object to be the CGI configuration, and have that configuration

assign the same type (CGI program) to every file, then you may have an arbitrary number of CGI directories without duplicating configuration information for each.

- By a wildcard expression. The Netsite Communications Server supports the use of wildcard expressions much like those used by the Unix shell to specify a set of pathnames which should be grouped into an object. Using this method, single files can be specified by giving their path, whole directories can be specified by giving the path followed by */**, and various other tricks (such as **.html*) can be used to construct a set of documents which should all have the same configuration information.

To control the behavior of the entire server, one named object is considered the root object. This object must contain all of the name translation directives for the server, and contains any global configuration changes you want to make.

The Contents of Objects

The Netsite server design breaks down the process of responding to an information request into the following steps:

- Authorization translation. Translate any HTTP user authorization information given by the client into a user and group. If necessary, decode the message to get the actual request.
- Name translation. Before anything else is done, a URL must be translated into a filesystem-dependent name or redirection URL.
- Path checks. Perform various tests on the resulting path. This is largely used to make sure that it's safe for the given client to retrieve the document.
- Object types. Determine the MIME type information for the given document. MIME types can be registered document types such as *text/html* and *image/gif*, or may 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.
- Service. Select an internal server function which should be used to send the result back to the client. This function may be a simple file blast, a CGI execution, or one of many other type-dependent functions.

- Log. Select a function or functions to be used to record information about the transaction which just finished.

These steps map directly to six of the configuration directives allowed in each object. There is a seventh configuration directive which controls how the server responds to the client when it encounters an error.

Format of Object Configuration Files

Object configuration files are comprised of lines of directives. There are seven directives:

- AuthTrans
- NameTrans
- PathCheck
- ObjectType
- Service
- AddLog
- Error

Each line has the following format:

```
Directive fn=function [parm1=value1] ... [parmN=valueN]
```

Note: Note that `fn=function` may appear anywhere among the other parameters.

Directive must be at the beginning of the line, with no leading white space. Lines starting with # are considered comments lines and are ignored. There may be an arbitrary amount of white space between each *parm=value* pair, but there must be no space between the parm, the equals sign and its value. You can put double quotes (") around the value.

Any line with leading white space is considered a continuation of the preceding line. For example:

```
PathCheck fn=grab-widget
   widget=square
```

is the same as:

```
PathCheck fn=grab-widget widget=square
```

If there is more than one instance of a particular directive in an object, the directives are applied in the order they appear in the file (top-down).

Knowing a bit about HTML can help you understand the underlying structure of object configuration files. In HTML, regions of text can be marked by having an opening anchor, the text, and then a closing anchor. Any text between the anchors is affected by the anchor.

Similarly, the object configuration file uses opening anchors and closing anchors to specify regions of configuration directives which are attributed to different objects.

Note: Note that the Object tag has two possible parameters: **name** and **ppath**, which correspond to creating a named object and specifying an object with wildcard expressions. You can use both if you want.

To get a better handle on this, here's a simple example configuration file:

```
<Object name=default>
NameTrans fn=document-root root=/usr/netsite
PathCheck fn=unix-uri-clean
ObjectType fn=type-by-extension
ObjectType fn=force-type type="text/plain"
Service fn="send-file"
</Object>
<Object ppath=/usr/netsite/local/*>
AddLog fn=common-log name=localaccess
</Object>
```

In this file, there are two objects. The first is named `default`, and the second applies to all files from the directory */usr/netsite/local*. The directives between the second set of `<Object>` and `</Object>` tags only apply to the specified directory, while the directives between the first set of Object tags apply to the root object and thus affect the whole server.

Access Control

Another use of HTML-like regions is to allow certain directives to be hidden to certain hosts, or to make those directives only apply to certain hosts. Other HTTP servers use a strict binary methodology for access control: a host is either forbidden or allowed to access a document. Netsite allows directives to be applied only to certain hosts, which allows for a very powerful set of customization options. For instance, by hiding an **AddLog** directive from certain hosts, you can exclude those hosts from your log files.

Note: Note that the Client tag takes two parameters: **dns** and **ip**. You can use both in one tag. The `/Client` tag can be placed after more than one directive (i.e. you can make the region larger than one directive).

To accomplish this, a region called `Client` is placed around directives within objects. For instance, let's tinker with the second object from the example above:

```
<Object ppath=/usr/netsite/local/*>
<Client dns=*.sgi.com>
AddLog fn=common-log name=localaccess
</Client>
</Object>
```

Now, the `localaccess` log will only contain accesses from hosts in the `sgi.com` domain. The **dns** and **ip** parameters to the `Client` tag are wildcard expressions telling the server to which clients to apply the directive. If you want to apply the directive to everybody except a certain host or IP address, use *~ as a prefix to your wildcard pattern.

A PathCheck function is provided which forbids access and duplicates the forbidden / not forbidden type of access control provided by other servers. This directive is then only applied to unwanted hosts. This can be more efficient because it doesn't require a full check each time.

Functions

Now that you know the basic format of the file, you need to become familiar with the server's functions and their parameters. Functions are divided into six classes, depending on the directive that should be used to apply them.

Note: It is not wise to use functions for one directive with another directive.

The following pages describe each directive, what it's used for, and then details each of the functions available of that class. The functions and directives described here include:

- AuthTrans Functions
- NameTrans Functions
- PathCheck Functions
- ObjectType Functions
- Service Functions
- AddLog Functions
- Error Directive

AuthTrans Functions

AuthTrans stands for Authorization Translation. Server resources can be protected such that accessing them requires the client to provide certain information about the user who is using that client. This HTTP user authorization information is encoded with some sort of simple security to prevent clients from authorizing themselves as a different user.

The server breaks the HTTP user authorization of client users into two steps: translating HTTP authorization information sent by the client, and requiring 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 which record HTTP user authorization information but do not require it.

If there is more than one AuthTrans directive in an object, all functions will be applied.

For each callable function, this section provides the following information: name and description, notes, required parameters, optional parameters and examples. The single AuthTrans function covered here is NCSA-Style Basic Authorization.

NCSA-Style Basic Authorization

Name and Description

basic-ncsa translates authorization information provided through the HTTP basic authorization scheme. This scheme uses user names and passwords which are sent in the clear across the network, making them vulnerable to snooping attacks.

Notes

This function is usually used in conjunction with the PathCheck function **require-auth**.

Required Parameters

auth-type gives the type of HTTP user authorization to be used. For this function, the setting should always be **basic**.

One of either **userfile** or **dbm** must be provided. Providing both yields undefined results.

userfile specifies the full pathname of the user database, in the NCSA httpd user file format. This format consists of `name:password` lines where password is encrypted.

dbm specifies the full path and base file name 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 an arbitrary number of all system users.

Optional Parameters

grpfile specifies the NCSA 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.

Examples

```
AuthTrans fn=basic-ncsa auth-type=basic dbm=/var/mc-httpd/userdb/rs

AuthTrans fn=basic-ncsa auth-type=basic userfile=/var/mc-httpd/.htpasswd
grpfile=/var/mc-httpd/.grpfile
```

NameTrans Functions

NameTrans stands for Name Translation. When people access your server, they will give URLs like:

```
http://yourserver.[yourdomain].[dom]/foo/bar
```

The */foo/bar* doesn't correspond directly to any full pathnames on your system. Instead, this is a *virtual path.*

The Netsite server uses NameTrans functions to translate this virtual path into the full pathname of a file or directory on your system. This is done to protect your system and prevent remote users from looking at your private files. If you are using the **Chroot** function to change your server root to the directory with your documents, then you can safely avoid using name translation.

NameTrans directives should appear in the root object, although you can put them elsewhere. If there is more than one NameTrans directive in an object, the server applies the functions until one succeeds and modifies the virtual path forming a full filesystem path.

For each callable function, this document provides the following information: name and description, notes, required parameters, optional parameters and examples. The NameTrans functions covered here include:

- Prefix to Directory
- User Public Information Directories
- Document Root
- Redirect
- Home Page

Prefix to Directory

Name and Description

pfx2dir looks for a certain directory prefix at the beginning of the virtual path the client is requesting. If it finds the prefix, it replaces the prefix with a real directory name.

Notes

The directory and the prefix you give should *not* have trailing slashes. Having them there will cause Not Found errors.

Required Parameters

from is the prefix to be mapped.

dir is the directory that prefix should be mapped to.

Optional Parameters

name gives a named object from which to derive configuration for this directory.

Examples

```
# Designate /sw/httpd/cgi-bin as a CGI directory
NameTrans fn=pfx2dir from=/cgi-bin dir=/sw/httpd/cgi-bin name=cgi

# Make all requests for /icons come from /var/mc-httpd/admin/icons
NameTrans fn=pfx2dir from=/icons dir=/var/mc-httpd/admin/icons
```

Users' Public Information Directories

Name and Description

unix-home is a NameTrans function which allows your system's users to provide information out of their home directories. This function allows you to designate a URL prefix which corresponds to user directories. Any request starting with this prefix then comes from a subdirectory of the user's home directory.

Notes

If you want to have the server scan the passwd file only once at startup, use the Init function **init-uhome**. This option saves wear and tear on your NIS server or local passwd file, but makes the server's resident size a bit larger.

You cannot use a **ppath** such as:

```
~user/public_html/*
```

to specify an object by wildcard expression. Instead, you must use:

```
/home/user/public_html/*
```

or whatever the full pathname is.

Required Parameters

from is the URL prefix to map to your users' directories. Previous servers used **/~** for this prefix.

subdir is the subdirectory of the users' directory which contains their documents.

pwfile is the full pathname of a *passwd* file to use, if you wish to use a file other than */etc/passwd* or the NIS database.

Optional Parameters

name gives a named object from which to derive configuration for this directory.

Examples

```
# Map /~user/foobar to ~user/public_html/foobar, get config from userhomes
NameTrans fn=unix-home prefix=/~ subdir=public_html name=userhomes

# Same thing, but let's use /u/ instead.
NameTrans fn=unix-home prefix=/u/ subdir=public_html name=userhomes
```

Document Root

Name and Description

document-root is a NameTrans function which specifies one directory to contain all of your documents. This directory will be prepended to the virtual path the client sends to form the full pathname of the file or directory to be used.

Notes

The directory you give should *not* have a trailing slash.

Required Parameters

root gives the directory which contains documents.

Optional Parameters

None.

Examples

```
# Requests such as /foobar/blatz should come from /d/netsite/foobar/blatz
NameTrans fn=document-root root=/d/netsite
```

Redirect

Name and Description

redirect is a NameTrans function which creates pointers to other URLs on your server. When a client accesses your server and gives a certain virtual path, they are told to use the URL you provide instead of the one they had.

Notes

None.

Required Parameters

from gives the virtual path prefix to be redirected.

At least one of **url** or **url-prefix** must be given.

url gives a single URL to redirect any request beginning with **from**.

url-prefix gives a URL prefix to redirect any request beginning with **from**. The prefix will be replaced by the URL, and anything following the prefix will be appended to the URL.

Optional Parameters

None.

Examples

```
# This server has been temporarily moved
NameTrans fn=redirect from=/ url-prefix=http://tmpserver/

# This directory moved to another server, so we map them to a moved document
NameTrans fn=redirect from=/popular-stuff url=http://bigserver/popular/wemoved.html
```

Home Page

Name and Description

home-page is a NameTrans function which sets your server's home page. When people access your server as:

```
http://yourserver.[yourdomain].[dom]
```

by default they will get an index file from the document root. You may specify another file with this function.

Notes

The file must exist on a local file system of the server machine.

Required Parameters

path is the new home page. If this is a partial path such as `welcome.html`, the client's request will be restarted as if it accessed the:

```
http://yourserver.[yourdomain].[dom]/welcome.html
```

If you specify a full path, the path you give will be used as the full file system path of the client's request.

Optional Parameters

None.

Examples

```
# Set my home page to /my/document/root/homepage.html
NameTrans fn=home-page path=homepage.html

# Set my home page to /d/netsite/welcome.html
NameTrans fn=home-page path=/d/netsite/welcome.html
```

PathCheck Functions

PathCheck directives check the full filesystem path which is returned after all of the NameTrans directives finish execution. The path is checked for such things as CGI path info and for things like `../` and `//` which are dangerous elements. Then any access restriction is applied.

If there is more than one PathCheck directive in an object, all of the directives are applied in the order they appear.

For each callable function, this document provides the following information: name and description, notes, required parameters, optional parameters and examples. The PathCheck functions covered here include:

- URI Cleaning for Unix Systems
- Find Index Files
- Require Authorization
- Deny the Existence Of Certain Paths
- Find Filesystem Links
- Find Path Information

URI Cleaning for IRIX Systems

Name and Description

unix-uri-clean is a PathCheck function. If a client ever sends a path with `../` in it, this can fool the file open system call into opening a document outside the document root. If a client sends a path with a `//` in it, then the file open system call allows that path to open a file, but any of your pathname-based access restrictions are not applied. This function will deny access to any client whose path has these elements in it.

Notes

If you plan to have scripts with extra path information, and that path information has `//` in it, be sure you call **find-pathinfo** before calling **unix-uri-clean**.

Required Parameters

None.

Optional Parameters

None.

Examples

```
PathCheck fn=unix-uri-clean
```

Find Index Files

Name and Description

find-index is a PathCheck function which determines if the requested path is a directory. If it is, then the function searches for a pre-written index file in the directory. If it finds one, it changes the path to point to the index file. If none is found, no action is taken, and later the server will most likely be asked to automatically generate a listing of the directory.

Notes

If the client was accessing a directory, and the virtual path did not end in a slash, then this function would redirect them to a new URL which has the trailing slash (/). This is done so that relative links in the HTML index are functional.

Required Parameters

index-names is a comma separated list of file names to look for. There should be no spaces unless they are part of the file names.

Optional Parameters

None.

Examples

```
# Look for /dir/index.html and /dir/home.html
PathCheck fn=find-index index-names=index.html,home.html
```

```
# Just look for index.html
PathCheck fn=find-index index-names=index.html
```

Require Authorization

Name and Description

require-auth is a PathCheck function which performs the second step of HTTP user authorization: denying access to unauthorized clients.

Notes

This function requires that an AuthTrans directive be executed prior to execution of **require-auth**.

Required Parameters

auth-type is the type of HTTP user authorization. Currently, the only valid type is basic.

realm is a short string which tells the client for which resource the server is requiring HTTP authorization.

Optional Parameters

auth-user specifies which users to allow. If missing, any valid user is allowed.

auth-group specifies which groups to allow. If missing, no group is required, and any valid group is allowed.

Examples

```
# Require group stalag-staff or users hogan, kinch
PathCheck fn=require-auth auth-type=basic realm="TurkishPlans"
 auth-group=stalag-staff auth users=(hogan|kinch)

# Just let anybody in who has been authorized
PathCheck fn=require-auth auth-type=basic realm="secret recipe"
```

Deny the Existence of Certain Paths

Name and Description

deny-existence is a PathCheck function which sends a Not Found error when a client tries to access a path. If this directive is protected by a Client region, then it performs access control.

Notes

Not Found is sent instead of Forbidden to help hide paths from unwanted clients.

Required Parameters

None.

Optional Parameters

path gives a wildcard expression of the path to check. Not specifying this parameter is equivalent to specifying "*****". Paths matching the expression you give will have their existence denied.

bong-msg is a file to send instead of the canned error message the server wants to send. This allows you to tell the client why it is unwanted.

Examples

```
# Deny existence of emacs auto-backup files
PathCheck fn=deny-existence path=*\~

# Make sure nobody except company employees see this document
<Client dns=*~*.sgi.com>
PathCheck fn=deny-existence bong-msg=/var/mc-httpd/go-away.html
</Client>
```

Find Filesystem Links

Name and Description

find-links is a PathCheck function which searches along the current path, looking for symbolic or hard links. If any are found, the server returns a Server Error and logs a message in the error log.

Notes

This function is only generally useful for directories you don't trust, such as users' home directories. It is debatable how much disabling links really helps safety, since the users can't access anything through the server that they can't already access (and copy) as themselves.

Required Parameters

disable is a character string specifying which links to disable. If the string contains **h**, then hard links are disabled. If the string contains **s**, soft links are disabled. If the string contains **o**, and the path is coming out of a user's home directory, symbolic links are allowed only if the target of the link is owned by that user.

dir is the directory at which to start checking. If this is the full pathname of the directory, then the current path will be checked against this one, and if the current path is not from this directory no action is taken. Link checks start at the path component which comes after the given directory. If **dir** is not a full path, link scans start at the first occurrence of the given string.

Optional Parameters

None.

Examples

```
# Directory maintained by foreign system, we don't trust it
PathCheck fn=find-links disable=sh dir=/foreign-dir

# Don't let our users use symlinks unless it's theirs. Start at subdir
# of their home to avoid automounter conflicts
PathCheck fn=find-links disable=so dir=public_html
```

Find Path Information

Name and Description

find-pathinfo is a PathCheck function which looks at a given path, and looks in the filesystem for the file. If it doesn't find the file, it attempts to find extra path information for CGI. Extra path information is information conveyed through extra directories following the file name being accessed. For example, in the path:

```
/myscript.cgi/foo/bar
```

/myscript.cgi is the file which exists, and */foo/bar* is path information. If **find-pathinfo** is unable to find a file, or finds a directory before it finds a file, no action is taken and the request is returned as Not Found.

Notes

None.

Required Parameters

None.

Optional Parameters

None.

Examples

```
# Scan for CGI path information in this directory.
PathCheck fn=find-pathinfo
```

ObjectType Functions

ObjectType directives determine the MIME type of the file being sent to the client. This type is usually sent back to the client to allow the client to readily decide what to do with it. MIME attributes currently sent are **type**, **encoding**, and **language**.

If there is more than one ObjectType directive in an object, all of 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 taken.

For each callable function, this document provides the following information: name and description, notes, required parameters, optional parameters and examples. The ObjectType functions covered here include:

- File Typing By File Name Extension
- File Typing By Wildcard Pattern
- Forcing File Types
- Server-Parsed HTML Hacks

File Typing By File Name Extension

Name and Description

type-by-extension is an ObjectType directive which uses file name extensions to determine information about a file. File name extensions are the things after the first period in a file name. In *foo.tar.gz*, *tar* and *gz* are the extensions. **type-by-extension** strips off extensions one at a time, and tries to determine a type, language, or encoding for each one.

Notes

If a name begins with a dot and the string after the dot matches a type, the file may be incorrectly flagged as a certain type.

It is truly regrettable that the most reliable and least error prone method of determining file types in “modern” filesystems is to use magical names.

Required Parameters

None.

Optional Parameters

None.

Examples

```
ObjectType fn=type-by-extension
```

File Typing By Wildcard Pattern

Name and Description

type-by-exp is an ObjectType function which tries to match the current path with a wildcard expression and, if it matches, the **type** information given as the function parameters is used.

Notes

None.

Required Parameters

exp is the wildcard expression of paths to which the information is applied.

Optional Parameters

type is the type to give any matching paths.

enc is the encoding to give any matching paths.

lang is the language to give any matching paths.

Examples

```
# Anything ending in .googol or with an extension .googol.something
# is of type application/googolplex
ObjectType fn=type-by-exp exp=*.googol($|.*) type=application/googolplex
```

Forcing File Types

Name and Description

force-type is an ObjectType function which just sets the **type** to the parameter you give.

Notes

This is used to implement a default type for the server.

Required Parameters

None.

Optional Parameters

type is the type to give any matching paths.

enc is the encoding to give any matching paths.

lang is the language to give any matching paths.

Examples

```
# Force anything that hasn't been typed at this point to be plaintext
ObjectType fn=force-type type=text/plain

# Force a default language for my documents
ObjectType fn=force-type language=en_US
```

Server-Parsed HTML Hacks

Name and Description

shtml-hacktype are provided for backward compatibility with server-side includes. The problem is that server-side includes require a different MIME type than HTML. This means that your parsed documents must have different file name extensions than your non-parsed ones. If this is a problem, this function is a solution. One solution is to have the server parse all HTML, which could get costly in terms of performance. The other

solution is to check for the execute bit on the file. If it's on, the file is parsed, otherwise, it isn't.

Notes

Netscape Communications Corporation does not recommend these solutions.

Required Parameters

None.

Optional Parameters

exec-hack, if present, tells the function to check to see if the exec bit is enabled for this file. If this parameter is not present, all files are marked as parsed.

Examples

```
# Parse every HTML file we have
ObjectType fn=shtml-hacktype

# Parse only those documents which have the execute bit on
ObjectType fn=shtml-hacktype exec-hack=true
```

Service Functions

Once the other directives have found a file to send, the Service class of functions actually takes care of sending the data and completing the request. For most files, the Service function simply sends the binary data back to the client untouched.

Service directives support all the following Optional Parameters, which help determine whether the directive will be used or not:

- **type** gives a wildcard expression of MIME types to which the directive is applied. The server defines several MIME types internally which are used only to select a Service function which will translate the internal type into a form presentable to the client.
- **method** specifies a wildcard expression of HTTP methods which the client must be using to have the directive applied. Valid HTTP methods are **GET**, **HEAD**, and **POST**.
- **query** specifies a wildcard expression of search queries which must be present for the directive to be executed.

If there is more than one Service directive in an object, the first applicable directive is executed and the rest are ignored.

For each callable function, this document provides the following information: name and description, notes, required parameters, optional parameters and examples. The Service functions covered here include:

- Send a Plain File
- Send an Error Message
- Append a Trailer To HTML Documents
- Execute a CGI Program
- Set a Default Query Handler
- Imagemap
- Simple Directory Indexing
- Fancy Directory Indexing
- Parse HTML (also known as server-side includes)

Send a Plain File

Name and Description

send-file sends the contents of a plain file back to the client.

Notes

If this function finds any extra path information, it flags the request as Not Found.

Required Parameters

None.

Optional Parameters

None.

Examples

```
# For non-internal types, accessed with GET or HEAD, send file back
# without touching its contents
Service type=*~magnus-internal/* method=(GET|HEAD) fn=send-file
```

Send an Error Message

Name and Description

send-error sends a certain HTML file back to the client regardless of the path the client was requesting. This is useful largely for error messages.

Notes

The file must be HTML.

Required Parameters

path specifies the full filesystem path of the HTML file to send.

Optional Parameters

None.

Examples

```
# Send back a message saying this resource has moved
Service fn=send-error path=/popular/service/we-moved.html

# Send a polite message saying we don't allow POSTing to normal files
Service fn=send-error path=/d/netsite/errors/no-post.html
```

Append a Trailer to HTML Documents

Name and Description

append-trailer appends text to the end of every HTML document from the current object. This is primarily useful for author information and copyright messages. The date of last modification for the file can be automatically included in the message.

Notes

This function pre-calculates the length of the document after the trailer is appended, allowing the last-modified and content-length HTTP headers to be passed back to the client. This is primarily useful to caches.

If extra path information is found, the request is flagged as Not Found.

Required Parameters

trailer is the text of the trailer to be appended to the documents. This text may contain HTML tags. The magical string `:LASTMOD:` is replaced by the last modification date of the file, using the time format specified by **timefmt**. If **timefmt** is not specified, no action will be taken.

Optional Parameters

timefmt is a time string in the **strftime** function format. See the manual pages for a description of the possible flags.

Examples

```
# Insert our copyright message at the end of every HTML document
Service type=text/html fn=append-trailer
trailer="<hr><img src=/logo.gif> Copyright 1994"

# Personalize my HTML documents
Service type=text/html fn=append-trailer timefmt="%D"
trailer="<hr><a href=/~rob/>Rob McCool</a>, last update :LASTMOD:"
```

Execute a CGI Program

Name and Description

send-cgi is a Service function which executes a file as a CGI program and sends the results to the client.

Notes

This documentation page assumes you already know about CGI. There's a good tutorial about CGI in Appendix C..

Required Parameters

None.

Optional Parameters

None.

Examples

```
# Execute every file in this object as a CGI program
Service fn=send-cgi

# Execute only files ending with .cgi as CGI programs
Service type=magnus-internal/cgi fn=send-cgi
```

Set a Default Query Handler

Name and Description

query-handler is a Service function which executes the CGI program you specify instead of referencing the path being requested.

Notes

This function's primary purpose is to support the obsolete ISINDEX tag. If possible, a FORM should be used instead.

Required Parameters

path is the full pathname of a CGI program to be executed.

Optional Parameters

None.

Examples

```
# Handle all searches with grep
Service query=* fn=query-handler path=/d/netsite/cgi/do-grep

# Handle all searches with searcher, give it path info to specify parameter
Service query=* fn=query-handler path=/d/netsite/cgi/search/parm1=faster
```

Imagemap

Name and Description

imagemap is a Service function which parses imagemap files. Map files specify ranges of coordinates as well as the URL that should be sent when the user clicks in one of them.

Notes

This page does not describe the format of a map file. There's a good tutorial about imagemaps in "What Is An Imagemap?" on page 91 in Appendix B.

Required Parameters

None.

Optional Parameters

None.

Examples

```
Service type=magnus-internal/imagemap method=(GET|HEAD) fn=imagemap
```

Simple Directory Indexing

Name and Description

index-simple takes a filesystem directory and produces a simple bulleted HTML list of the contents.

Notes

If **index-simple** creates a link to a directory (because one of the items found is a sub-directory), the function does not provide the server with enough information to know such a link exists, so it does not append the URL with a trailing slash. In this situation, when the user clicks on the link, the client is redirected to the sub-directory. In other words, it takes 2 connections to follow a directory link with the **index-simple** function; with the **index-common** function it just takes 1 connection (see “Fancy Directory Indexing” on page 188).

Required Parameters

None.

Optional Parameters

None.

Examples

```
Service type=magnus-internal/directory fn=index-simple
```

Fancy Directory Indexing

Name and Description

index-common creates a listing of a directory in the common directory indexing format implemented by the CERN and NCSA HTTP servers. This format contains quite a bit of information, looks fancier than simple indexing, and references icons.

Notes

This form of indexing may be resource-intensive, especially with scan HTML titles turned on.

The behavior of this function can be tweaked with the **cindex-init** Init function.

Required Parameters

None.

Optional Parameters

header is a file to prepend to the indexing which introduces the contents of the directory. If you specify *filename* for this parameter, the server looks for *filename.html* first and incorporates that as HTML if found. Otherwise, it incorporates the file name as plain text.

readme is a file to append to the indexing which gives more information about the contents of the directory. If you specify *filename* for this parameter, the server looks for *filename.html* first and incorporates that as HTML if found. Otherwise, it incorporates the file name as plain text.

Examples

```
# Look for HEADER and README for top and bottom of indexing pages
Service type=magnus-internal/directory method=(GET|HEAD) fn=index-common
header=HEADER readme=README
```

Parse HTML

Name and Description

parse-html (also known as server-side includes) parses an HTML document, scanning for embedded server directives. These server directives are executed to provide certain information only the server has, or to include the contents of other files.

Notes

This page assumes you are familiar with the parsed HTML directives.

Parsing lots of HTML documents can be costly in terms of performance.

Required Parameters

None.

Optional Parameters

opts specifies the options for parsing. **no-exec** is the only current option, which disables the use of the exec directive.

Examples

```
# Parse SHTML documents, allowing use of the exec directive.
Service type=magnus-internal/parsed-html method=(GET|HEAD) fn=parse-html
```

AddLog Functions

After the request completes and the server has stopped talking to the client, there is still work that the server needs to do. The server records information about every access clients make, and records information about the client making the request as well.

If there is more than one AddLog directive in an object, all are applied.

For each callable function, this document provides the following information: name and description, notes, required parameters, optional parameters and examples. The AddLog functions covered here include:

- Log In the Common Format
- Record the Client Software Name

Log In the Common Format

Name and Description

common-log is an AddLog function which records request-specific data in the common log format used by most HTTP servers. There are a number of freely available statistics generators for this format.

Notes

None.

Required Parameters

None.

Optional Parameters

name gives the name of a log file, which must have been given as a parameter to the **init-clf** Init function. If no name is given, **global** is assumed.

iponly, if present, instructs the server to skip looking up the hostname of the remote client, and records the IP address instead.

Examples

```
# Log all accesses to the central log file
AddLog fn=common-log

# Log non-local accesses to another log file
<Client ip=*~198.93.9[2345].*>
AddLog fn=common-log name=nonlocal
</Client>
```

Record the Client Software Name

Name and Description

record-useragent is an AddLog function which records the IP address of each client followed by the User-Agent HTTP header they provided. If none was provided, the server records "(none given)". The User-Agent header field tells the server what version of Netscape Navigator (or another HTTP-based client) is in use.

Notes

None.

Required Parameters

None.

Optional Parameters

name gives the name of a log file, which must have been given as a parameter to the **init-clf** Init function. If no name is given, **global** is assumed.

Examples

```
# I'm curious what software people are using, so I can see how many
# forms-capable clients use my server.
AddLog fn=record-useragent name=useragents
```

Error Directive

At any time during a request, a certain number of conditions may occur which cause the server to stop fulfilling a request and return an error to the client. When this happens, the server can send a short HTML page to the client telling them in very general terms about the error.

In order to make error handling more friendly, the Netsite Commerce Server allows you to intercept certain errors and send a file of your choosing instead of the server's canned error message.

The following is a list of errors which are returned by the server. Each line has the 3-digit HTTP code which designates the error, followed by a short reason.

- `401 Unauthorized`

  The files in this object are protected (see the AuthTrans function **basic-ncsa** and the PathCheck function **require-auth**). The server requires HTTP user authorization to allow access, and the client either provided none or its HTTP authorization was insufficient.

- `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.

- `404 Not Found`

  The client asked for a filesystem path which doesn't exist, or which the server has been instructed to tell the client that it doesn't exist. If you use access control, changing the response to this error can let you nicely tell people to go away.

- `500 Server Error`

  Server errors mean that an error has occurred within the server which prevents it from finishing the request. Server errors can happen because of misconfigurations, CGI programs exiting early or otherwise failing, or machine resources such as swap space being exhausted.

Most Service functions can be called from Error directives. To decide when to apply an Error directive, there are two optional parameters given to Service functions being used from Error directives:

reason gives one of the above reason strings in lowercase (such as `forbidden` or `not found`).

code gives a 3-digit error code such as `404` or `500`.

See “Service Functions” on page 182 to determine which functions you can call from an Error directive.

Netsite Technical Specifications

Netsite Technical Specifications

- All HTTP-based clients supported. Serves HTML documents
- Backward compatibility with NCSA httpd. Speaks industry-standard HTTP 1.0 protocol
- CGI 1.1 compliant
- DNS and IP address based access control
- HTTP 1.0-based access authorization
- Client accesses logged in common log file format
- Server restarts or log files rotate with SIGHUP
- Multiple log files which can restrict logging from certain hosts
- Simplified imagemap setup
- User public information directories (IRIX users)
- Allow user databases other than /etc/passwd or NIS map
- Can be pre-loaded to save network or disk bandwidth
- Common indexing format for directories
- Customized error messages
- Simplified installation, configuration and maintenance through intuitive online forms
- Increased load capacity and system load limitation through creation of a configurable number of recyclable server processes
- Server configuration control per-directory, per-file, by shell wildcard pattern or using templates
- Streamlined server-parsed HTML
- Symbolic link searches friendly to automounter

- Document root and virtual-to-physical mappings
- Ability to disable symbolic links and hard links
- MIME typing through filename extensions (including compression)
- DBM hashed file format used for high capacity user database
- Supports UNIX-based operating systems from: Digital Equipment Corp. (OSF/1 2.0), Hewlett Packard (HP-UX 9.03), IBM RS/6000 (AIX 3.2.5), Silicon Graphics (IRIX 5.3), Sun (Solaris 2.3 & 2.4; SunOS 4.1.3), 386/486/Intel Pentium PCs (BSDI 1.1)

Glossary

CGI

Common Gateway Interface—an interface for external programs to "talk" to the HTTP server. Programs that are written to use CGI are called CGI programs or CGI scripts. CGI programs do things like handle forms or perform output parsing not normally done by the server. For more information about CGI scan the Contents page at the beginning of this guide and, specifically, read Appendix C.

common log file format

The common logfile format is the format used by the server for entering information into the access and error logs. The format was originally developed through a joint effort between NCSA and CERN.

DNS

Domain Name System. The system used by machines on a network to associate standard IP addresses (such as 198.93.93.10) with hostnames (such as www.mcom.com). Machines normally get this translation information from a DNS server, or look it up in tables maintained on their systems.

DNS alias

A DNS alias is a hostname that the DNS server knows points to a different host—specifically, a DNS CNAME record. For example, `www.[yourdomain].[dom]` may be an *alias* that points to a real machine called `wilma.[yourdomain].[dom]` where the server currently exists. When somebody uses Netscape Navigator or some other network navigation software to access the server, the only URL they need is:

`http://www.[yourdomain].[dom]`

Later, should you move the server to another machine called `fred.[yourdomain].[dom]`, you simply update the DNS CNAME record to reflect the server's new location and user's can continue to use:

`http://www.[yourdomain].[dom]`

but need not know or be aware that the server has actually moved.

document root

A directory on the server machine that contains the files, images and data you want to present to users accessing the server.

EMACS

A Unix text editing program—also used to read e-mail and news. EMACS originally stood for Editing MACroS (the baroque interpretation of this acronym is “Escape Meta Alt Control Shift”).

fancy indexing

Fancy indexing provides more information than simple indexing. Fancy indexing displays a list of contents by name with file size, last modification date, and an icon reflecting file type. Because of this, fancy indexes may take longer than simple indexes for the client to load.

file extension

The last section of a file’s name, usually preceded by the last period in the name. 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 does not have the same file type as a music file. Often identified by the file’s extension (for example, *.gif*, *.wav*, *.html*.).

GIF

A cross-platform image format originally created by Compuserve. The acronym stands for Graphics Interchange Format. GIF files are usually much smaller in terms of file size than they would be in their native platform-specific format. GIF is one of the most common interchange formats. GIF images are readily viewable on UNIX, Microsoft Windows™ and Apple Macintosh® systems using the appropriate software.

home page

A document that exists on the server and acts as a catalog or entry point for the server’s contents. The location of this document is defined within the server’s configuration files.

hostname

A name for a machine of the form `machine.domain.dom`, which is translated

to an IP address. For example: `www.mcom.com` is the machine `www` in the subdomain `mcom` with domain `com`.

HTML

The acronym stands for Hypertext Markup Language and is the document formatting language used by Netscape Navigator. HTML controls the format of text, the positioning of graphics and forms input items, and the navigable links found on pages which are viewed with HTML-compatible network navigation software.

HTTP

Hypertext Transport Protocol. The method by which information is exchanged on a network between HTTP servers and clients.

httpd

An abbreviation for HTTP daemon, which means a program which serves information using the HTTP protocol. The Netsite Commerce Server is often called an httpd.

imagemap

A process which makes areas of an image active, allowing users to navigate and obtain information by clicking the different regions of the image with a mouse. Imagemap can also refer to a CGI program call “imagemap” which is used to handle imagemap functionality in other httpd implementations.

inittab

Often called */etc/inittab* because of its location, it is a file which describes programs that need to be restarted if they should ever die (to provide continual service).

IP address

Internet Protocol address—a set of numbers, separated by dots, which specify the actual location of a machine on the network.

ISINDEX

Documents can often use a network navigator’s capabilities to accept a search string and send it to the server to access a searchable index without using forms. In order to use ISINDEX, you must create a query handler.

ISMAP

`ISMAP` is an extension to the `IMG SRC` tag used in an HTML document to tell the server that the named image is an imagemap.

MIME

Multi-Purpose Internet Mail Extensions. This is an emerging standard for multimedia e-mail and messaging.

NIS

Network Information System—a system of programs and data files that Unix machines can 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—a research organization at the University of Illinois at Urbana-Champaign, where the original research prototype of Netscape was designed and produced.

password file

A file on Unix machines that stores Unix user login names, passwords and user ID numbers, among other things. Known as */etc/passwd* in the case of the Unix password file, because of where it is kept.

public information directories

Directories not inside the document root that are in a Unix user's home directory (or directories) which are under the user's control.

RAM

Random Access Memory. The physical semiconductor-based memory in a computer.

rc.local

Often called */etc/rc.local* because of its location, it is a file which describes programs that need to be run at machine startup.

realm

A term used in HTTP access authorization that helps the user identify what part of the system is asking for an HTTP user name and password. For

example, the realm in the Netsite Server Manager is “Server Administrator.”

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 be able to find the new location transparently.

resource

Any file, directory or program that the server can access and send to a client that asks for it.

root

The most privileged user available on Unix machines. Has complete access privileges to all files on the local machine.

ScriptAlias

The way that NCSA httpd did some of its configuration, including directory remapping and CGI activation.

server daemon

The server daemon ***is*** the server. The server daemon is a process which, once executed, listens for and accepts requests from clients.

server root

A directory on the server machine dedicated to holding the server configuration, maintenance and information files.

simple index

The opposite of fancy indexing—this type of directory listing displays only the names of the files without any graphical elements.

strftime

A function that converts a date and a time to a string. Used by the server when appending trailers. **strftime** has a special format language for the date and time that the server can use in a trailer to illustrate a file’s last modified date.

superuser

A level of Unix system privileges equal to those of the *root* user.

sym-links

Abbreviation for symbolic links—these are a type of redirection used by the Unix operating system. They allow you to create a pointer from one part of your file system to an existing file or directory on another part of the file system.

telnet

A protocol for two machines on the network to be connected to each other and to support terminal emulation for remote login.

top

A program that exists on some Unix systems which can show the current state of system resource usage.

top-level domain authority

The highest category of hostname 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).

uid

A unique number associated with each Unix user on a machine.

URL

Uniform Resource Locator—the address 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.mcom.com/index.html
```

Index

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

W