**NCSA HTTPd Server**
Software Development Group
National Center for Supercomputing Applications
University of Illinois at Urbana−Champaign
605 E. Springfield, Champaign IL 61820
httpd@ncsa.uiuc.edu

## Copyright (C) 1995, Board of Trustees of the University of Illinois

**NCSA HTTPd** software, both binary and source (hereafter, Software) is copyrighted by The Board of Trustees of the University of Illinois (UI), and ownership remains with the UI.

The UI grants you (hereafter, Licensee) a license to use the Software for academic, research and internal business purposes only, without a fee. Licensee may distribute the binary and source code (if released) to third parties provided that the copyright notice and this statement appears on all copies and that no charge is associated with such copies.

Licensee may make derivative works. However, if Licensee distributes any derivative work based on or derived from the Software, then Licensee will (1) notify **NCSA** regarding its distributing of the derivative work, and (2) clearly notify users that such derivative work is a modified version and not the original **NCSA HTTPd** Server software distributed by the UI by including a statement such as the following:

> "Portions developed at the National Center for Supercomputing Applications at the University of Illinois at Urbana−Champaign."

Any Licensee wishing to make commercial use of the Software should contact the UI, c/o **NCSA**, to negotiate an appropriate license for such commercial use. Commercial use includes (1) integration of all or part of the source code into a product for sale or license by or on behalf of Licensee to third parties, or (2) distribution of the binary code or source code to third parties that need it to utilize a commercial product sold or licensed by or on behalf of Licensee.

Any commercial company wishing to use the software as their commercial World Wide Web server and are not redistributing the software need not commercially license the software but can use it free of charge.

UI MAKES NO REPRESENTATIONS ABOUT THE SUITABILITY OF THIS SOFTWARE FOR ANY PURPOSE. IT IS PROVIDED "AS IS" WITHOUT EXPRESS OR IMPLIED WARRANTY. THE UI SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY THE USERS OF THIS SOFTWARE.

By using or copying this Software, Licensee agrees to abide by the copyright law and all other applicable laws of the U.S. including, but not limited to, export control laws, and the terms of this license. UI shall have the right to terminate this license immediately by written notice upon Licensee’s breach of, or non−compliance with, any of its terms. Licensee may be held legally responsible for any copyright infringement that is caused or encouraged by Licensee’s failure to abide by the terms of this license.





Back to NCSA HTTPd Overview[1]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / Last Modified 08−01−95*

[1] Overview.html
[2] FeedBack.html

# NCSA HTTPd

**NCSA HTTPd** is an HTTP/1.0[1] compatible server for making hypertext and other documents available to Web browsers. Please read this copyright notice[2].

The current version is 1.5. If you are using an earlier version, these upgrade notes[3] will be helpful. We also have an FAQ[4] on **NCSA HTTPd**.

---

Documentation is available on the following topics:

**Why NCSA HTTPd?**[5]
: A brief introduction to the server and its capabilities.

**Installation Instructions and Setup**[6]
: Downloading[7] – Compiling[8] – Configuring[9]
Step by step instructions for getting the server running.
...or use the **NEW** SingleClick NCSA HTTPd Downloader![10] Great for first–time users...

**Administrating the server**[11]
: How to get the server to meet your needs.

**Tutorials**[12]
: Directory Indexing[13] – Server Side Includes[14] – CGI Configuration[15]
Imagemap Usage[16] – VirtualHost Configuration[17]
In depth tutorials on various aspects of server configuration.

**Acknowledgements**[18]
: The people who have made **NCSA HTTPd** what it is today.

**The HTTPd Library**[19]
: Everything you need to know about writing a server.

---

You may also perform a keyword search[20] on this documentation.

Search:

Any questions or bug reports regarding **NCSA HTTPd** should go to `httpd@ncsa.uiuc.edu`.[21]

---

[1] http://www.w3.org/hypertext/WWW/Protocols/HTTP1.0/draft–ietf–http–spec.html
[2] COPYRIGHT.html
[3] Upgrade.html
[4] FAQ.html
[5] info/Features.html
[6] setup/Install.html
[7] setup/Download.html
[8] setup/Compilation.html
[9] setup/Configure.html
[10] setup/SingleClick.html
[11] setup/admin/Overview.html
[12] tutorials/
[13] tutorials/index–tutorial.html
[14] tutorials/includes.html
[15] tutorials/cgi.html
[16] tutorials/imagemapping.html
[17] tutorials/VirtualHost.html
[18] acknowledgement.html
[19] Library.html
[20] /cgi–bin/wais.pl
[21] FeedBack.html





Return to the **NCSA HTTPd** homepage.[22]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[23] / Last Modified 08–01–95*

[22] ../index.html
[23] FeedBack.html

# Frequently Asked Questions



---

### 1.1) What is NCSA HTTPd?

NCSA HTTPd[18] is a program to serve information, much in the same way that NCSA Mosaic[19] is a program to browse information in the World Wide Web.[20] From the Client−Server viewpoint, NCSA HTTPd is the Server to the Browser Client.

HTTPd stands for HyperText Transfer Protocol (HTTP) Daemon. HTTP[21] is a protocol with the lightness and speed necessary for a distributed collaborative hypermedia information system.

1 #whatis
2 #how
3 #cgi
4 #winhttpd
5 #forms
6 #imagemaps
7 #users
8 #wais
9 #ssi
10 #env
11 #more
12 #logfiles
13 #server
14 #gid
15 #hidden
16 #multiple
17 #pgppem
18 Overview.html
19 http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html
20 http://www.w3.org/
21 http://www.w3.org/hypertext/WWW/Protocols/HTTP/HTTP2.html





### 1.2) How do I install NCSA HTTPd?

Read the documentation on NCSA HTTPd Installation.[22]

### 1.3) What is CGI?

CGI stands for Common Gateway Interface. You can read an introduction[23] and an overview.[24]

### 1.4) Where can I find HTTPd for Microsoft Windows/NT ™?

For information on Microsoft Windows™ HTTPd, please read this page.[25]
For information on Microsoft Windows NT ™HTTPd , please read this page[26].
**Note: Since these server packages were NOT developed by NCSA, we are not responsible for them. Our Technical Support Personnel don't know anything about them, and will not answer any questions about them.**

---

### 2.1) How do I create and process forms?

First, read a tutorial on creating forms.[27] Then, in order to handle the output from your forms, read how CGI supports forms.[28]

**Note:** Not all sites allow all users to run CGI scripts on their servers. In order to run, a CGI script must be in a ScriptAlias directory or the server must allow CGI scripts anywhere by the CGI Magic Type convention. See the CGI Configuration Tutorial[29] for more information.

### 2.2) How do I setup a clickable imagemap?

Read the documentation on the imagemap script[30] and a tutorial[31] on graphical information maps.

### 2.3) How do I manage users for authentication?

Read the documentation on managing users[32] and a tutorial[33] on user authentication.

### 2.4) How can I perform keyword searches?

You can use WAIS as a back−end search engine. Herein, a tutorial[34] on WAIS and HTTP integration.

### 2.5) How do I set up server−side includes?

You can use server−side includes, to provide data to clients on−the−fly. For example,

22 setup/Install.html
23 /cgi/intro.html
24 /cgi/
25 http://www.city.net/win−httpd/
26 http://emwac.ed.ac.uk/html/internet_toolchest/https/contents.htm
27 http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill−out−forms/overview.html
28 /cgi/forms.html
29 tutorials/cgi.html
30 setup/admin/Imagemap.html
31 tutorials/imagemapping.html
32 setup/admin/UserManagement.html
33 tutorials/user.html
34 tutorials/wais.html

the current date or the size or last modification date for a file. Check out a tutorial[35] on server−side includes.

### 2.6) How to add environment variables?

You can add environment variables when you use POST method.
For more infomation, please read CGI Environment Variables[36]

### 2.7) What else can I do?

Read the complete documentation on server administration[37] and some provided step−by−step tutorials[38].

---

### 3.1) My logfiles are not updated anymore...

This is probably because you moved them or erased them without restarting a standalone daemon. For more information, read the Logfile Management[39] documentation.

### 3.2) When I submit a form, it gave me 501 error...

The error message says:

```
We are sorry to be unable to perform the method POST to
non−script at this time.
```

This means that the server did not recognize the referenced URL (the URL defined as the ACTION in a HTML Form) as a CGI script. For the server to recognize the URL as a CGI script, it must either reside in a ScriptAlias[40] directory or your site must allow CGI scripts anywhere. For more information, see the CGI Configuration Tutorial[41].

**Note:** Some versions of the NCSA HTTPd server request that you mail NCSA if you would like this feature supported in new versions of NCSA HTTPd. Nine times out of Ten we receive requests to support this feature, when in fact it was a misconfiguration which caused the error message. Please verify with the owner of the script/form before asking us to fix it.

### 3.3) Can't start server with "gid error"/"setgroups"

This may be caused by the values in User directive[42] and Group directive[43]. The default values for these two directives are all set to −1. These values are not valid on some systems. In particular, HP−UX does not support −1 or −2 as valid groups. By default, HP−UX defines nogroup as −2, but the server cannot use this. Please consult with your system administrator to find the valid values.

---

[35] tutorials/includes.html
[36] cgi/env.html#headers
[37] setup/admin/Overview.html
[38] tutorials/
[39] setup/admin/Logs.html
[40] setup/srm/ScriptAlias.html
[41] tutorials/cgi.html
[42] setup/httpd/User.html
[43] setup/httpd/Group.html





### 4.1) Can I have hidden variables in forms?

Actually this is handled by the browser (e.g. NCSA Mosaic). In any case, there is an example with hidden variables[44] in the documentation.

### 4.2) Can I have multiple submit buttons?

This is also handled by your browser. NCSA Mosaic for X as of v2.5 has support for multiple submit buttons. See an example[45] to learn how to use them.

### 4.3) What is PGP/PEM encryption?

PGP and PEM are programs to allow you and a second party to communicate in a way which does not allow third parties to read your messages. For more information, there is a PGP/PEM FAQ[46].

---

Return to the Overview[47]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[48] / Last Modified 9−08−95*

[44] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill−out−forms/example−13.html
[45] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/fill−out−forms/example−14.html
[46] PEMPGP.html
[47] Overview.html
[48] FeedBack.html

# NCSA HTTPd Library

### Everything you Need to know to write an HTTP Server

- Internet Drafts, Standards, and RFCs
    - Hypertext Transfer Protocol –– HTTP/1.0[1]
    - W3C HTTP Documents[2]
    - RFC 1775 – To Be "On" the Internet[3]
    - RFC 1521 – MIME[4]
    - RFC 959 – FTP[5]
    - RFC 977 – NNTP[6]
    - RFC 1036 – Standard for Interchange of USENET Messages[7]
    - RFC 822 – EMAIL[8]
    - Gopher[9]
    - Z39.50[10]
- Common Gateway Interface
    - CGI 1.1/NCSA[11]
    - CGI Current Practice[12]
    - W3 Overview of CGI[13]
- Server Side Imagemaps
    - NCSA Imagemap Tutorial[14]
- Security
    - Basic Authentication
        - W3C Access Authorization Overview[15]
    - MD5 Digest Authentication
        - RFC 1321 on MD5 Digest Algorithm[16]
        - Draft IETF specification for MD5 Digest Authentication[17]
        - Spyglass' Efforts on WWW Security[18]

---

Return to NCSA HTTPd 1.5 Home Page[19]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[20] / 9–27–95*

[1] http://www.w3.org/hypertext/WWW/Protocols/HTTP1.0/draft–ietf–http–spec.html
[2] http://www.w3.org/hypertext/WWW/Protocols/
[3] http://sdg.ncsa.uiuc.edu/~mag/work/specs/RFCs/rfc1775.txt
[4] http://ds.internic.net/rfc/rfc1521.txt
[5] http://www.w3.org/hypertext/WWW/Protocols/rfc959/Overview.html
[6] http://www.w3.org/hypertext/WWW/Protocols/rfc977/rfc977.html
[7] http://www.w3.org/hypertext/WWW/Protocols/rfc1036/rfc1036.html
[8] http://www.w3.org/hypertext/WWW/Protocols/rfc822.txt
[9] http://www.w3.org/hypertext/Gopher/gopher_protocol/protocol.rtf
[10] ftp://sunsite.unc.edu/pub/Z39.50/Standards_documents
[11] http://hoohoo.ncsa.uiuc.edu/cgi/
[12] http://www.ast.cam.ac.uk/%7Edrtr/cgi.html
[13] http://www.w3.org/pub/WWW/CGI/
[14] tutorials/imagemapping.html
[15] http://www.w3.org/hypertext/WWW/AccessAuthorization/Overview.html
[16] http://ds.internic.net/rfc/rfc1321.txt
[17] http://ds.internic.net/internet–drafts/draft–ietf–http–digest–aa–01.txt
[18] http://www.spyglass.com/six/developers_tech_doc2.html
[19] index.html
[20] FeedBack.html

# NCSA HTTPd 1.4 Pre−Forking Model

I originally looked into the performance problems when we started having problems with www.acm.uiuc.edu[1] which hosts Rob's Multimedia Lab[2] on its RS/6000 360. The decision we made was to have a single process pre−fork to spawn several children. The parent selects() on the main socket, and after accepting connections, it passes it to one of the spawned children.

NCSA HTTPd 1.4 supports both SYSV and BSD file descriptor passing, as well as a NO_PASS mode for systems which don't support file descriptor passing. One such popular system is Linux, the free unix operating system. Several Linux developers are rumoured to be working on fixing this deficiency.

I later had a brain storm about possibly having multiple processes fork and then all sit on accept() on the main socket, but this behaviour is not documented in the socket description, and different OS's handle this in different fashions. We decided to stick with the file descriptor passing method already implemented.

Some Notes: Not all of the previously supported systems were available to test file descriptor passing. The default behaviour is to compile these systems without it. It is known to be supported under SunOS4, Solaris2[3], Irix[4], HP−UX[5], AIX[6], Ultrix, OSF/1,[7] NeXT[8], NetBSD, and SCO SVR3.2[9]. In order to compile with file descriptor passing, you must define FD_BSD or FD_SYSV in the configuration section for your system in `src/httpd.h`. Even if your system defines SYSV instead of BSD, you will probably use FD_BSD (As most of the above systems do.) If you do add this feature to another system, please mail the information to httpd@ncsa.uiuc.edu[10].

Also, if you are a larger server, you need to watch out for the limit on the number of open files allowed per process, and the number of processes allowed per user. The server maintains a pipe to each child to pass the file descriptors, so you need at least `4+MaxServers` file descriptors allowed, and more to allow servers above MaxServers. Systems like SunOS 4, under the GENERIC compile, have a maximum of 64. Some systems, like Solaris 2 allow the system administrator to set the maximum. The maximum number of processes per user is a problem since HTTPd setuids to the user you specify in httpd.conf. If user nobody is only allowed 40 processes (the default under AIX 3.2.x), and you have more than 40 children (or around) the child cannot run any CGI scripts. AIX allows you to change this value via SMIT.

*Brandon*

Return to the Upgrade page[11].

[1] http://www.acm.uiuc.edu/
[2] http://www.acm.uiuc.edu/rml/
[3] http://www.sun.com/
[4] http://www.sgi.com/
[5] http://www.hp.com/
[6] http://www.ibm.com/
[7] http://www.dec.com/
[8] http://www.next.com/
[9] http://www.sco.com/
[10] FeedBack.html
[11] Upgrade.html





*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[12] / Last Modified 9−08−95*

[12] FeedBack.html

# Using PGP/PEM encryption

**Note:** *As of NCSA HTTPd 1.4.1, support for PEM/PGP encryption was removed in order to bring NCSA in compliance with the International Treaty on Arms Reduction to which the United States of America is a signatory. We hope to have an improved version available with NCSA HTTPd 1.5 from an export controlled server.*

This document is formatted roughly like a FAQ. Here are the section headings:



---

## What is PGP/PEM encryption?

PGP and PEM are programs to allow you and a second party to communicate in a way which does not allow third parties to read them, and which certify that the person who sent the message is really who they claim they are.

PGP and PEM both use RSA encryption. The U.S. government has strict export controls over foreign use of this technology, so people outside the U.S. may have a difficult time finding programs which perform the encryption.

---

## How are they used in HTTP?

We have implemented a system by which NCSA Mosaic and NCSA HTTPd call external programs which encrypt and decrypt their communications and thus provide secure communications between the server and the client and ensure that a user is who he/she says they are. This system currently has hooks for PEM encryption as well as PGP encryption. As interest in this area grows, more will most likely be added.

---

## What do I need to use it?

You will need a working copy of either Pretty Good Privacy or RIPEM to begin

[1] #whatis
[2] #how
[3] #needs
[4] #keydist
[5] #holes
[6] #server
[7] #client
[8] #protocol





with. You should be familiar with the program and have generated your own public/private key pair.

You should be able to use the TIS/PEM program with the PEM authorization type. I haven't tried it. This tutorial is written assuming that you are using RIPEM.

---

## How do the keys get distributed?

Currently, we have implemented this protocol with PEM and PGP using local key files on the server side, and on the client side with PEM using finger to retrieve the server's public key.

As you can tell, parties who wish to use Mosaic and HTTPd with PEM or PGP encryption will need to communicate beforehand and find a tamper−proof way to exchange their public keys.

---

## How bulletproof is it?

Pioneers get shot full of arrows. This work is currently in the experimental stages and thus may have some problems that I have overlooked. There aren't any that I can see but I've been looking at it for a long time. There may be some quirks in the additions to Mosaic and HTTPd as well. In particular, error recovery is not always as helpful as it could be.

The only known problem that I know about is that the messages are currently not timestamped. This means that a malicious user could record your encrypted message with a packet sniffer and repeat it back to the server ad nauseum. Although they would not be able to read the reply, if the request was something you were being charged for, you may have a large bill to pay by the time they're through.

---

## Installation: The Server Side

First, you must compile HTTPd with CFLAGS set to −DPEM_AUTH. This will enable the PEM and PGP authentication directives.

Next, look in the `support/auth` directory. This directory contains your encryption and decryption scripts, as well as bins for your remote users' public keys.

Edit ripem−dec, ripem−enc, pgp−enc, and pgp−dec and follow the instructions therein. You need not set up PGP if you don't plan to use it, and same with RIPEM.

Now, edit your server configuration file, usually conf/httpd.conf. You will want to add three new directives for PGP and three new directives for PEM depending on which you plan to use (or both).

```
PEMEncryptCmd /usr/local/etc/httpd/auth/ripem-enc
PGPEncryptCmd /usr/local/etc/httpd/auth/pgp-enc
```

These directives change what executables HTTPd will look for when it is trying to encrypt its reply to a client. Edit the pathnames to taste.

```
PEMDecryptCmd /usr/local/etc/httpd/auth/ripem-dec
PGPDecryptCmd /usr/local/etc/httpd/auth/pgp-dec
```

These directives change what executables HTTPd will look for when it is trying to decrypt the client's request.

```
PEMServerEntity webmaster@foobar.org
PGPServerEntity webmaster@foobar.org
```

These directives set your entity name. This should be the same as the name you place on the public/private keys you generate for your server. If you make these directives different than the key names, your server and its client will become hopelessly confused.

You are now ready to protect directories of your server with this authorization scheme.

For a directory you want to protect, you should first set its AuthType. Use `AuthType PGP` for a directory you are protecting with PGP and `AuthType PEM` for a directory you are protecting with PEM.

The require directive accepts key names as its arguments. The AuthGroupFile directive is valid as well, to create groups of keys. A full example:

```
AuthType PEM
AuthGroupFile /httpd/.htgroup-pem
Options None
<Limit GET>
require user robm@ncsa.uiuc.edu
require group pemusers
```

Let's say /httpd/.htgroup–pem reads:

pemusers: pls@ncsa.uiuc.edu In this case, this directory will be protected with PEM encryption and will require that only users robm@ncsa.uiuc.edu and pls@ncsa.uiuc.edu be allowed to access that directory.

---

## Installation: the Client Side

First, get a copy of Mosaic/X 2.2. If it hasn't been released yet, be patient. Compile it with –DPEM_AUTH to enable PEM/PGP authentication.

Follow the instructions in each of the scripts in the auth subdirectory to customize them to your setup.

There are six new X resources which have been defined for PEM/PGP authentication. They are:

```
Mosaic*pemEncrypt: /X11/robm/Mosaic/auth/ripem-enc
Mosaic*pemDecrypt: /X11/robm/Mosaic/auth/ripem-dec
Mosaic*pemEntity: robm@ncsa.uiuc.edu
Mosaic*pgpEncrypt: /X11/robm/Mosaic/auth/pgp-enc
Mosaic*pgpDecrypt: /X11/robm/Mosaic/auth/pgp-dec
Mosaic*pgpEntity: robm@ncsa.uiuc.edu
```

You should change the Encrypt and Decrypt entries to reflect where you are going to install your encryption and decryption crypts. You should change the Entity lines to





the key name you have given the server maintainers for yourself. If you don't, bad things will happen.

---

## What does the protocol look like?

This protocol is almost word–for–word a copy of Tony Sander's RIPEM based scheme, generalized a little. Below, wherever you see PEM you can replace it with PGP and get the same thing.

**Client:**

```
GET /docs/protected.html HTTP/1.0
UserAgent: Mosaic/X 2.2
```

**Server:**

```
HTTP/1.0 401 Unauthorized
WWW-Authenticate: PEM entity="webmaster@hoohoo.ncsa.uiuc.edu"
Server: NCSA/1.1
```

**Client:**

```
GET / HTTP/1.0
Authorization: PEM entity="robm@ncsa.uiuc.edu"
Content-type: application/x-www-pem-request

--- BEGIN PRIVACY-ENHANCED MESSAGE ---
this is the real request, encrypted
--- END PRIVACY-ENHANCED MESSAGE ---
```

**Server:**

```
HTTP/1.0 200 OK
Content-type: application/x-www-pem-reply

--- BEGIN PRIVACY-ENHANCED MESSAGE ---
this is the real reply, encrypted
--- END PRIVACY-ENHANCED MESSAGE ---
```

That's it.

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[9] / Last Modified 9–08–95*

[9] FeedBack.html

# Upgrading NCSA HTTPd

## HTTPd 1.5

Primary Focus on Security

- Multiple Directory Indexes[1]
- Redirects from .htaccess files[2]
- RedirectPermanent[3]
- MD5 authentication[4]
- Kerberos v4 authentication[5]
- Kerberos v5 authentication[6]
- HTTP 1.0 KeepAlive[7]
- DBM .htpasswd/.htgroup files[8]
- Enhanced Access Control[9]
- Multihome/Virtual Interface support[10]
- Multiple RefererIgnore
- Fixed problems with −HUP restart on some systems

## HTTPd 1.4.2

*Maintenance Release*

- Few more bug fixes:
  - minor access bug
  - Patch to avoid cfg_getline loop on user error
  - fix AddType application/x−httpd−cgi cgi in .htaccess files
  - Don't log 2 transactions on errors handled via ErrorDocument
  - Don't add DOCUMENT_ROOT environment variable twice
  - Fix inetd support in set_group_privs()
- Support for AIX 4.1.x and HP/Apollo DomainOS

## HTTPd 1.4.1

*Maintenance Release*

- The few bug fixes, most notably:
  - If you set StartServers/MaxServers to 0, httpd might crash
  - Hopefully, the Content−type with 302 scripts problem
  - segmentation faults on local redirects

1 features−1.5.html#DirectoryIndex
2 features−1.5.html#.htaccess−Redirect
3 features−1.5.html#RedirectPermanent
4 features−1.5.html#MD5_AUTH
5 features−1.5.html#KERB_AUTH
6 features−1.5.html#KERB_AUTH
7 features−1.5.html#KeepAlive
8 features−1.5.html#DBM
9 features−1.5.html#SEC_ENHANCE
10 features−1.5.html#MULTI_HOME





  - Really changed to REDIRECT_ on ErrorDocument env vars
- Removed PEM/PGP hooks so as to comply with ITAR (International Trade in Arms Regulations), at the recommendation of the NSA (these hooks will be available on an export−controlled ftp server, to be set up soon)
- Added SECURE_LOG define to close possible security hole of leaving the logfiles open for the CGI scripts

## HTTPd 1.4

The primary focus of release 1.4 of the NCSA HTTPd server is to improve performance[11].

### Changes from HTTPd 1.3R

- Two new configuration directives have been added: *StartServers*[12], *MaxServers*[13]
  Previously, the HTTPd server forked with each new request To reduce this overhead the NCSA HTTPd was modified to support a hunt group. A pool of servers (of size _StartServers_) is created initially. As the load increase additional servers are started as needed to the configured maximum (_MaxServers_). These servers remain in the server pool. If additional requests are received, new servers are forked which answer the request, and die in a similar fashion to previous standalone versions of the code. If you don't want to have a pool of servers at all, you can set up StartServers and MaxServers to 0. In this way, HTTPd's behavior will be similar to previous versions of the code in standalone form. For more Information[14]

- Introduced a new mail script: /cgi−bin/mail
  Associated file: /cgi−bin/mail.list
  This PERL script uses the sendmail binary and an authorized list of recipients to generate an email form for the users on your Web server. It should fulfill the mail needs of Web users everywhere currently asking about custom email scripts and mailto: for sending form results: If called directly, it will output a blank form to fill in, and sends the values to itself for mailing. If referenced in the ACTION line of a form, it will forward the FORM INPUT to the named recipient.
  Configuration documentation is provided at the top of the script. Official documentation will be available soon.

- Previously each forked child made initialization calls to retrieve information available to the parent. This information is now retrieved by the parent processed once and passed to all child processes.

- Additional performance and memory management improvements internal to the server have been made.

- Several string copies have been modified to prevent buffer overflows. This addresses the problem reported in CERT Advisory CA−95:04. The CERN suggestion of increasing the buffer size was not implemented because we took a different approach. We do not recommend blindly increasing the buffer

11 http://www.ncsa.uiuc.edu/InformationServers/Performance/V1.4/report.html
12 setup/httpd/StartServer.html
13 setup/httpd/MaxServer.html
14 FilePass.html

sizes because this can cause excessive swapping and performance degredation.

- The NCSA HTTPd 1.4 supports modifiable error messages for the following errors using a new server resource map directive, *ErrorDocument*[15].
Modifiable error messages REDIRECT 302 BAD_REQUEST 400 AUTH_REQUIRED 401 FORBIDDEN 403 NOT_FOUND 404 SERVER_ERROR 500 NOT_IMPLEMENTED 501
- Two new logs are supported, an agent log and a referral log. The agent log lists the identity of the requesting client. The referral log lists the URL containing the link to the requested document.
Three new configuration directives have been added: *AgentLog*[16], *RefererLog*[17], and *RefererIgnore*[18].

## HTTPd 1.3R

Fixed the vulnerablity described in Cert Advisory CA−95:04[19]. For more information, please read the description[20].

## HTTPd 1.3

HTTPd 1.3 is a maintenance release designed to fix some of the bugs which were introduced in HTTPd 1.2.

**Changes from HTTPd 1.2**

- Now compiles cleanly under A/UX and Solaris.
- Directory indexing bugs repaired, new options to suppress last modified, size, and description columns (thanks Tanmoy.)
- Wildcard based access control should work this time.
- Fixed core dump related to recursive parsed document include.
- Fixed bug by which HTTPd would not follow a symbolic link it should have.
- Fixed file typing (no longer case sensitive like in 1.2.)
- Parent directory escaped in indexes.
- IdentityCheck should work again.
- LAST_MODIFIED and #flastmod are now local zone.
- Time related functions work with NeXT now.
- Fixed spelling error in 500 error page.

[15] setup/srm/ErrorDocument.html
[16] setup/HTTPd/AgentLog.html
[17] setup/HTTPd/RefererLog.html
[18] setup/HTTPd/RefererIgnore.html
[19] /security/CA_95:04/CA_95:04.txt
[20] /security/CA_95:04/ncsa_patch.html





- Server will log proper timeout message for non−DNS hosts.
- Added x−bit hack for people to use the x−bit of an HTML file to determine if it is parsed or not.
- Added compile defines so people can customize the amount of DNS the server performs (for speed).

## HTTPd 1.2

HTTPd 1.2 makes some significant changes over HTTPd 1.1. These changes make its "look and feel" a bit different than 1.1.

**New features**

- The CGI[21] interface is now revised to version 1.1. See the changes[22] that were made.
- A new logfile format was agreed upon which should simplify the lives of the authors of logfile analyzers. Quickly, the format is:

  ```
  host rfc931 authuser [DD/Mon/YYYY:hh:mm:ss [+/−]HHMM] "request" status byt
  ```

  Host = hostname

  rfc931 = RFC931 user name if IdentityCheck active, − if not.

  authuser = HTTP/1.0 authenticated user, − if none.

  DD/Mon/YYY:hh:mm:ss [+/−]HHMM: Local time of the request with timezone offset from GMT at end.

  "request" = the request as sent by the client

  status = the HTTP/1.0 status code from this transaction

  bytes = the count of the bytes sent in this transaction, not including the header. If not applicable, this will be a − character.
- CGI[23] scripts are now allowed anywhere. See this tutorial[24] on the new setup (and how it relates to the old setup), and how to disable this feature[25] in places you don't want it.
- The server side includes[26] interface has been completely rewritten. See this tutorial[27] to learn how to convert your old INC SRV documents, and to learn how to take advantage of the new features.
- There is a new access control option[28] to disable symbolic links only if the owner of the pointer is not the same as the owner of that which is pointed to.

[21] /cgi/
[22] /cgi/10to11.html
[23] /cgi/
[24] tutorials/cgi.html
[25] tutorials/security.html
[26] info/Includes.html
[27] tutorials/includes.html
[28] setup/access/Options.html

This means your users can have symbolic links to things they own, but not to dangerous things like /etc.

- A new access control method called mutual–failure[29] has been added. This method is a bit unorthodox, but allows you to allow hosts from one domain while excluding certain hosts (such as public access machines) from that domain.
- Wildcard expressions[30] are now allowed in various areas of server configuration, to allow patterns to be specified. This is most useful in the Directory[31] directive.
- Directory indexing has been revamped. It looks much different[32] and I've written a short tutorial[33] on how to set it up.
- Access Control Files now allow the indexing directives as well as the DefaultType directive.
- require user now allows quotes for PGP usernames with spaces.
- Server now explicitly kills CGI scripts when the client aborts.
- Server now verifies the DNS hostname it gets from the IP number to prevent PTR spoofs.
- Support for 304 and If–modified–since.

**Bug fixes**

All of the known bugs in 1.1 have now been fixed. Now it's time to find the ones I introduce with 1.2.

- Fixed problem running scripts in ServerMode inetd under IRIX.
- Fixed bad port problem under OSF/1.
- Inserted missing return statement for the IdentityCheck directive.
- Fixed problem whereby errors would stop being logged after a restart.
- Fixed 256 character limitation on CGI URLs.

---

## HTTPd 1.1

HTTPd 1.1 should plug right in if you already have httpd 1.0.

- Introduced experimental PEM/PGP based encrypted user authentication. See this overview[34] to read about trying it out.
- Improved directory indexing. See the new directives in srm.conf[35] to use it.
- Cleaned up error output and fixed horrible output when server–side include

[29] setup/access/order.html
[30] setup/MatchExp.html
[31] setup/access/Directory.html
[32] info/star_trek/
[33] tutorials/index–tutorial.html
[34] PEMPGP.html
[35] setup/srm/Overview.html





error occurred
- Fixed slight bug in buffering code
- Directory indexing for user–supported directories fixed
- Retrieval of user supported directory with no trailing slash issues redirect again
- Now supports RFC931 identd for logging purposes. See the IdentityCheck[36] directive.
- stderr for scripts and server side includes now sent to error_log
- Fixed bug in NCSA POST script code which would cause Location: to be ignored for local files
- Removed misfeature wherein Location: url's were being escaped by the server
- Args to INC SRVURL escaped to avoid unpleasant surprises
- Location: /cgi–bin/foo?arg now works
- HEAD only for CGI scripts now ignores body put out by stupid scripts

---

## HTTPd 1.0

HTTPd 1.0 is fairly similar to HTTPd 1.0a5.

**Changes which directly require configuration changes**

- CGI script interface introduced
- ScriptAlias now used for CGI scripts only, access your NCSA scripts with OldScriptAlias
- Fixed AddType directive
- The NCSA scripts are no longer included, CGI replacements are now included

**Other changes**

- NCSA POST scripts now have REMOTE_HOST set
- Added AddEncoding directive to srm.conf and .htaccess
- error_log now logs all failed accesses and the reason for failure
- Time headers are now RFC822 compliant
- You can now alias or disable user–supported directories
- Added a require directive to allow you to allow any valid user
- I/O is now buffered into chunks for speed
- Redirect no longer supported from .htaccess files.
- Added –v and –f command line flags

---

## HTTPd 1.0a5

- Fixed horrible bug in 1.0a4

---

## HTTPd 1.0a4

- Introduced user authentication (Basic scheme)
- Multiple script directories now supported
- Fixed bug in http_request.c related to clients not sending a method
- Fixed bug in http_request.c which caused DirectoryIndex to be ignored and

[36] setup/httpd/IdentityCheck.html

index.html to be used anyway
- Added overrule for mime types in srm.conf and access config. so the user doesn't have to edit mime.types
- Requests for directories which do not come with a trailing slash and will return a prewritten index are now redirected to the same URL with a trailing slash
- The HEAD method is now supported
- Fixed a getline call which was hardcoded to 1024 instead of HUGE_STRING_LEN
- Fixed problem in which a user could GET /~user/../../../ and gain access outside of DocumentRoot
- httpd.conf split into two files: a server configuration and a document configuration file
- Introduced Redirect directive which allows you to redirect clients to documents' new locations
- Introduced htpasswd program for managing user authentication
- Fixed problems in http_access.c where certain .htaccess files were ignored and some symlinks were being followed when they shouldn't have
- Distribution now places config files as foo.conf−dist instead of foo.conf so it doesn't write over your old files
- Config files now case insensitive
- MIME header scans are now case insensitive per the RFC
- Added server config. diretive Timeout to control the timeouts when listening to or writing to a client
- Script execution now uses execl() instead of system() to eliminiate extra tmp files and to allow system() from scripts.
- New document config. directive DefaultType for type of files which cannot be typed from filename
- Script arguments now not touched: Added support program to decode them for shell scripts
- Added support for the post method of accessing forms
- Added support for "extra path" in script name
- Fixed so that an un−executable script now returns FORBIDDEN instead of blank page
- Spaces in content type returned from scripts no longer included, so include files now work from imagerect script.
- Thanks to Arjan de Vet, script execution no longer uses system() or popen(), reducing chroot requirements as well as allowing scripts to use system().
- Added DefaultType directive to define what type a document with an unknown or missing extension should be
- Added Timeout directive to allow servers to choose the length of time a client is given to send a request and to recieve the data.
- Added preliminary support for POST scripts, or more importantly, support for scripts which interpret the new form submission methods.
- Scripts now do all of their own URL decoding, which can be done either with provided C code, or with the support program `unescape`
- Scripts may now have extra path information after the script name, so allowing filters
- Improved Imagerect Script
- Allowed non−URL location returns from script to allow access control, user authentication, and typing (great idea, Ari).
- Returning content−length and last−modified headers





## HTTPd 1.0a3

- .htaccess files now affect subdirectories
- Added option FollowSymLinks for security
- Fixed MAJOR bug in init_mime function
- Enhanced Makefile and httpd.h for people compiling httpd themselves
- Added env. vars DOCUMENT_ROOT and SERVER_NAME for scripts

## HTTPd 1.0a2

- Introduced support for HTTP/1.0 3xx redirection from server scripts
- Introduced per−directory access by host and options control
- Added flags for BSD 4.2 compatibility (no killpg and no setsid)
- Enhanced documentation
- Introduced support for server restart through SIGHUP
- Fixed bug which caused system not to return port when HTTPd is done
- Fixed bug which caused server to go into infinite spin with GET /htbin
- Bug in directory indexing repaired (thanks to neal@ctd.comsat.com).
- Added timeout while sending data to client so as to alleviate zombie servers laying around when the client drops.

## HTTPd 1.0a1

- Introduced server scripts
- Server scripts for a variety of UNIX commands like archie added
- Server−side forms handling scripts added
- Image rectangling scripts added
- Introduced more extensible configuration file format
- Server now replies to HTTP/1.0 clients in HTTP/1.0 format
- Introduced MIME data type determination on the fly
- Improved logging of client access as well as server errors
- Major internal reworking
- Added timeout for clients that disconnect
- Added server include files

## HTTPd 0.5

- Introduced standalone support
- Final release for gopher support and experimental group annotation support
- No reported outstanding bugs

## HTTPd 0.4beta

- Introduced experimental group annotation support
- Introduced access logging
- Introduced HTTP/1.0 tolerance
- Handling of full hostnames added
- Bug fixed with index.html not being recognized without a trailing slash

- Fixed bug with extraneous character being added to the end of the data stream
- Fixed bug handling directory URL's with trailing slashes
- Made a little more friendly to Ultrix's brain death

---

## HTTPd 0.3beta

- Introduced drop−in gopher support
- Fixed security hole
- Not being able to find the config file does not send back a path

---

## HTTPd 0.2beta

- Two security holes fixed
- Double slashes in index of root fixed
- Not being able to find error files no longer hangs the server

---

Return to Overview[37]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[38]/ Last Modified 08−01−95*

[37] Overview.html
[38] FeedBack.html

# NCSA HTTPd Version History

This is very techinical, but details exactly what changed between releases of HTTPd. The list is in reverse chronological order.

## 1.1

- Introduced experimental PEM/PGP based encrypted user authentication
- Improved directory indexing
- Cleaned up error output and fixed horrible output when server−side include error occurred
- Fixed slight bug in buffering code
- Directory indexing for user−supported directories fixed
- Retrieval of user supported directory with no trailing slash issues redirect again
- Now supports RFC931 identd for logging purposes
- stderr for scripts and server side includes now sent to error_log
- Fixed bug in NCSA script code which would cause Location: to be ignored for local files
- Removed misfeature wherein Location: url's were being escaped by the server
- Args to INC SRVURL escaped to avoid unpleasant surprises
- Location: /cgi−bin/foo?arg now works
- HEAD only for CGI scripts now ignores body put out by stupid scripts

## 1.0

- CGI script interface introduced
- Fixed bug in AddType
- NCSA POST scripts now have REMOTE_HOST set
- Output now buffered in 1K chunks

## 1.0a5

- Fixed horrible bug in 1.0a4

## 1.0a4

- Introduced user authentication (Basic scheme)
- Multiple script directories now supported
- Fixed bug in http_request.c related to clients not sending a method
- Fixed bug in http_request.c which caused DirectoryIndex to be ignored and index.html to be used anyway
- Added overrule for mime types in srm.conf and access config. so the user doesn't have to edit mime.types
- Requests for directories which do not come with a trailing slash and will return a prewritten index are now redirected to the same URL with a trailing slash
- The HEAD method is now supported
- Fixed a getline call which was hardcoded to 1024 instead of HUGE_STRING_LEN
- Fixed problem in which a user could GET /~user/../../../ and gain access





outside of DocumentRoot
- httpd.conf split into two files: a server configuration and a document configuration file
- Introduced Redirect directive which allows you to redirect clients to documents' new locations
- Introduced htpasswd program for managing user authentication
- Fixed problems in http_access.c where certain .htaccess files were ignored and some symlinks were being followed when they shouldn't have
- Distribution now places config files as foo.conf−dist instead of foo.conf so it doesn't write over your old files
- Config files now case insensitive
- MIME header scans are now case insensitive per the RFC
- Added server config. diretive Timeout to control the timeouts when listening to or writing to a client
- Script execution now uses execl() instead of system() to eliminiate extra tmp files and to allow system() from scripts.
- New document config. directive DefaultType for type of files which cannot be typed from filename
- Script arguments now not touched: Added support program to decode them for shell scripts
- Added support for the post method of accessing forms
- Added support for "extra path" in script name
- Fixed so that an un−executable script now returns FORBIDDEN instead of blank page
- Spaces in content type returned from scripts no longer included, so include files now work from imagerect script.
- Thanks to Arjan de Vet, script execution no longer uses system() or popen(), reducing chroot requirements as well as allowing scripts to use system().
- Added DefaultType directive to define what type a document with an unknown or missing extension should be
- Added Timeout directive to allow servers to choose the length of time a client is given to send a request and to recieve the data.
- Added preliminary support for POST scripts, or more importantly, support for scripts which interpret the new form submission methods.
- Scripts now do all of their own URL decoding, which can be done either with provided C code, or with the support program `unescape`
- Scripts may now have extra path information after the script name, so allowing filters
- Improved Imagerect Script
- Allowed non−URL location returns from script to allow access control, user authentication, and typing (great idea, Ari).
- Returning content−length and last−modified headers

## 1.0a3

- .htaccess files now affect subdirectories
- Added option FollowSymLinks for security
- Fixed MAJOR bug in init_mime function
- Enhanced Makefile and httpd.h for people compiling HTTPd themselves
- Added env. vars DOCUMENT_ROOT and SERVER_NAME for scripts

## 1.0a2

- Introduced support for HTTP/1.0 3xx redirection from server scripts
- Introduced per−directory access by host and options control
- Added flags for BSD 4.2 compatibility (no killpg and no setsid)
- Enhanced documentation
- Introduced support for server restart through SIGHUP
- Fixed bug which caused system not to return port when HTTPd is done
- Fixed bug which caused server to go into infinite spin with GET /htbin
- Bug in directory indexing repaired (thanks to neal@ctd.comsat.com).
- Added timeout while sending data to client so as to alleviate zombie servers laying around when the client drops.

---

## 1.0a1

- Introduced server scripts
- Server scripts for a variety of UNIX commands like archie added
- Server−side forms handling scripts added
- Image rectangling scripts added
- Introduced more extensible configuration file format
- Server now replies to HTTP/1.0 clients in HTTP/1.0 format
- Introduced MIME data type determination on the fly
- Improved logging of client access as well as server errors
- Major internal reworking
- Added timeout for clients that disconnect
- Added server include files

---

## Version 0.5

- Introduced standalone support
- Final release for gopher support and experimental group annotation support
- No reported outstanding bugs

---

## Beta Version 0.4

- Introduced experimental group annotation support
- Introduced access logging
- Introduced HTTP/1.0 tolerance
- Handling of full hostnames added
- Bug fixed with index.html not being recognized without a trailing slash
- Fixed bug with extraneous character being added to the end of the data stream
- Fixed bug handling directory URL's with trailing slashes
- Made a little more friendly to Ultrix's brain death

---

## Beta Version 0.3

- Introduced drop−in gopher support
- Fixed security hole





- Not being able to find the config file does not send back a path

---

## Beta version 0.2

- Two security holes fixed
- Double slashes in index of root fixed
- Not being able to find error files no longer hangs the server

---

Return to the Overview[1].

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / Last Modified 9−11−95*

[1] Overview.html
[2] FeedBack.html

# Features of NCSA HTTPd 1.5

- Multiple Directory Indexes[1]
- Redirects from .htaccess files[2]
- RedirectPermanent[3]
- MD5 authentication[4]
- Kerberos v4 authentication[5]
- Kerberos v5 authentication[6]
- HTTP 1.0 KeepAlive[7]
- DBM .htpasswd/.htgroup files[8]
- Enhanced Access Control[9]
- Multihome/Virtual Interface support[10]
- Parent User directive (?)
- Multiple RefererIgnore

---

## Multiple Directory Indexes

NCSA HTTPd now accepts a list of index files that it will check for on directory URL references. No more hacks to enable CGI scripts or Server Side Includes as Directory Indexes.

*Documentation*[11]

## Redirects from .htaccess files

NCSA HTTPd now accepts Redirect directives in .htaccess files allowing for users to handle their own redirects.

*Documentation*[12]

## RedirectPermanent

NCSA HTTPd now accepts the RedirectPermanent (and RedirectTemp) directives where appropriate to return different status messages to clients which support them. The Redirect directive is supported for backward compatibility, and is equivalent to RedirectTemp.

*Documentation*[13]

[1] #DirectoryIndex
[2] #.htaccess–Redirect
[3] #RedirectPermanent
[4] #MD5_AUTH
[5] #KERB_AUTH
[6] #KERB_AUTH
[7] #KeepAlive
[8] #DBM
[9] #SEC_ENHANCE
[10] #MULTI_HOME
[11] howto/DirectoryIndex.html
[12] howto/Redirect.html
[13] howto/Redirect.html





## MD5 Authentication

NCSA HTTPd now supports the MD5 Basic Authentication scheme as proposed by Spyglass[14]. This allows for a more secure authentication as the passwords are not sent across the connection. There is even a timestamp to limit replay attacks.

*Documentation*[15]

## Kerberos v4 and v5 Authentication

NCSA HTTPd now supports Kerberos Authentication. This allows web client users to authenticate themselves to httpd by sending a Kerberos ticket as part of the HTTP request. The server also authenticates itself to the client when it sends back a successful response. This can be used for more flexible access control and authorization checking.

*Documentation*[16]

## HTTP 1.0 KeepAlive

NCSA HTTPd now supports the KeepAlive HTTP directive, allowing multiple requests to be handled in a single connection. On some verification tests, we found a speed increase of up to 30% with a KeepAlive capable client like NCSA Mosaic.

*Documentation*[17]

## DBM Support

NCSA HTTPd now supports DBM format files for password and group information. This provides significantly faster user lookup times for access controlled information with a large user list.

*Documentation*[18]

## Enhanced Access Control

NCSA HTTPd now supports an enhanced access control format, which is backward compatible with the existing format. With the new format, you can control access to directories with AND and OR booleans on restrictions. This would allow, for example, restricting access to a particular domain or a valid user.

*Documentation*[19]

## Multihome/Virtual Interface support

NCSA HTTPd now supports the ability to respond differently to different IP

[14] http://www.spyglass.com/
[15] howto/md5_auth.html
[16] howto/kerberos.html
[17] howto/KeepAlive.html
[18] howto/dbm_support.html
[19] howto/access_control.html

addresses. While similar to the famed APB patches, NCSA HTTPd takes it a step further, making most of the httpd.conf,srm.conf and PEM/PGP hooks configurable per host.

*Documentation*[20]

---

Back to NCSA HTTPd 1.5 Upgrade Notes[21]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[22]/ Last Modified 08−01−95*

[20] howto/multihome.html
[21] Upgrade.html
[22] FeedBack.html

# Why NCSA HTTPd?

Here is a list of the server's features. Follow the links to read about them and see demonstrations of their use.

---

**Low impact**
: Small, for ease of portability and security.
: Fast[1], to handle the load on the ever–expanding WWW.

**Compatibility**
: Designed to support HTTP/1.0 as implemented on the WWW
: Can also handle older browsers using HTTP/0.9

**Directory indexes**
: The user can create HTML indexes under a list of names (*see the DirectoryIndex[2] directive*) or the server can generate an index dynamically (see the Star Trek *example*[3])

**User directories**
: Easily allow your users access to the publishing end of the WWW, with enough security to keep them from harming your system.

**CGI[4] Server scripts[5]**
: Customize your server to execute searches and handle HTML forms.

**Security[6]**
: Limit access to your server by directory.
: Support for HTTP/1.0 Basic Authentication, MD5 Digest Authentication, and Kerberos versions 4 and 5.

**Server Side Includes[7]**
: Include the output of commands or other files in your HTML documents.

---

If you are interested, here is a version history[8] of past releases.

---

Return to the overview[9].

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / Last Modified 09–11–95*

1 http://www.ncsa.uiuc.edu/InformationServers/Performance/V1.4/report.html
2 ../setup/srm/DirectoryIndex.html
3 star_trek/
4 /cgi/
5 Scripts.html
6 Security.html
7 Includes.html
8 ../Upgrade.html
9 ../Overview.html
10 ../FeedBack.html

# Server Side Includes

NCSA HTTPd allows you to include the output of commands or other files in your HTML documents.

For instance, the current date is Tue Dec 5 21:52:40 CST 1995 .

At the end of each document, I'd like to include my signature but don't want to have to change every document I wrote when my home document moves. So, I can use a server include tag to include it, like this:

Signature for NCSA HTTPd Development Team:

**NCSA HTTPd Development Team**
*email: httpd@ncsa.uiuc.edu*
URL: http://hoohoo.ncsa.uiuc.edu/

Note:
Having the server parse documents is a double edged sword. It can be costly for heavily loaded servers to perform parsing of files while sending them. Further, it can be considered a security risk to have average users executing commands as the server's User[1]. If you disable the exec option, this danger is mitigated, but the performance issue remains. You should consider these items carefully before activating server–side includes on your server.

Return to the features list[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified 9–11–95*

1 ../setup/httpd/User.html
2 Features.html
3 ../FeedBack.html

# Server Scripts

**NCSA HTTPd** allows you to write server scripts, which are executable programs which the server runs to generate documents on the fly. They are easy to write and can be written in your favorite language, whether it be C, PERL, or even the Bourne shell.

---

These scripts can be used for a number of purposes:

### HTML forms

If you are using a browser which supports HTML forms, like **NCSA Mosaic**[1], you can create scripts to handle form requests.

If you are familiar with the PH system, you can look a person up in the phone book[2]. Try looking up Brandon Long by name.

### Simple HTML indexes

Most browsers support the HTML <ISINDEX> flag. You can use server scripts to respond to them.

For example, what is hoohoo's load average[3] right now?

Let's try a simple finger gateway[4]. Try looking for blong.

How about looking something up in archie[5]?

### Image mapping

If you are using a browser which supports it, you've probably noticed that image mapping is a fun feature. **NCSA HTTPd** lets you create your own maps.

Below is a picture of some ray traced fish. Try clicking on different areas of the picture and see what it does.

---

[1] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html
[2] /cgi−bin/phf
[3] /cgi−bin/uptime
[4] /cgi−bin/finger
[5] /cgi−bin/archie





[6]

---

Return to the features list[7].

---

**NCSA HTTPd** *Development Team / httpd@ncsa.uiuc.edu*[8] */ Last Modified 9−11−95*

[6] fish.map
[7] Features.html
[8] ../FeedBack.html

# Security

**NCSA HTTPd** allows you to control who can access different document trees on your server. It also allows you to define which server functions are carried out in different document trees.

Note that the following methods of access control can work independently, or be combined.

---

### Host Filtering

Host filtering is used to limit document trees to certain machines, most likely local machines or those directly involved in a project.

For example, I have some documents[1] which are for NCSA only. If you are not from a machine in ncsa.uiuc.edu, you can't access them.

I have another directory[2] that I don't want NCSA people to read, since they can access it elsewhere.

### User Authentication

If your users are using a browser that supports user authentication, like **NCSA Mosaic**[3], you can have the users authenticate themselves with a user name and a password before they are able to access protected documents.

For an example, let's create three users.

- `blong` password `greatone`
- `dpape` password `conehead`
- `swetland` password `bonehead`

I have protected this directory[4] so that only `blong` can access it.

I have protected this directory[5] so that only people in the Mosaic X developers group can access it. That means `dpape` and `swetland`.

I have protected this directory so that only `blong` can access it, and only from `void.ncsa.uiuc.edu`.

---

Return to the features list.[6]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[7] / Last Modified 9-11-95*

---

[1] ncsaonly/
[2] nonncsa/
[3] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html
[4] robonly/
[5] xmosdonly/
[6] Features.html
[7] ../FeedBack.html

# User Supported Directories

NCSA httpd allows you to let users make their own directories available for HTTP access, without your having to create links to them. Of course, you can limit what the users can do from their directories for security reasons.

---

Return to Access Overview[1]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / Last Modified 11−27−95*

[1] http://hoohoo.ncsa.uiuc.edu/docs/setup/access/Overview.html
[2] mailto:httpd@ncsa.uiuc.edu

## Star trek pictures

These are some fun pictures that people have digitized from the various Star Trek movies and TV shows.

| Name | Last modified | Size | Description |
|---|---|---|---|
| Parent Directory[1] | 27-Nov-95 16:38 | - | |
| birdprey.jpg[2] | 18-Jul-95 20:02 | 75K | A Klingon Bird of Prey |
| ent_rel.jpg[3] | 18-Jul-95 20:02 | 78K | The Enterprise and Reli> |
| enterds9.jpg[4] | 18-Jul-95 20:02 | 30K | Deep Space 9 |
| foo.txt[5] | 18-Jul-95 20:02 | 0K | |
| ncc1701.jpg[6] | 18-Jul-95 20:02 | 50K | |

Back to the Features of NCSA HTTPd 1.5[7]

Back to Indexing Options[8]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[9] / Last Modified 8−07−95*

[1] /docs/info/
[2] birdprey.jpg
[3] ent_rel.jpg
[4] enterds9.jpg
[5] foo.txt
[6] ncc1701.jpg
[7] ../Features.html
[8] ../../setup/srm/Overview.html
[9] mailto:httpd@ncsa.uiuc.edu

# Installation Instructions

Installing **NCSA HTTPd** can be broken down into these basic steps:

- Downloading the NCSA HTTPd Server[1]
- Finding HTTPd a Good Home[2]
- Configuring HTTPd[3]
- Selecting Scripts[4]
- Starting NCSA HTTPd[5]

---

## Downloading the NCSA HTTPd Server

If you already have a copy of the current release of **NCSA HTTPd 1.5**, you may skip to the next step[6].

**NCSA HTTPd** is available as precompiled distributions[7] and binaries[8] for systems we have access to.

There is also a new SingleClick[9] Download and Configure for easy access to **NCSA HTTPd**.

If you **already** have **NCSA HTTPd 1.4** running, you may update the HTTPd binary without the new support files.

*Note: There are several new support files for the new authorization types, as well as examples of many of the other new features in the configuration files, so you might want to retrieve the support files anyways.* A step−by−step guide[10] on how to update the old server is available.

If your system is not on this list, or if you feel more comfortable doing so, you must compile a binary[11].

---

## Finding HTTPd a Good Home

At this stage, you should move HTTPd's control files from the directory you have been working in to the directory you intend to use as ServerRoot[12] in the server configuration file.

Unless you have redefined their locations in the server configuration file, you will have to move the following files and directories into ServerRoot[13]:

[1] #Download
[2] #Move
[3] #Configure
[4] #Select
[5] #Start
[6] #Configure
[7] PreCompiled.html
[8] PreExec.html
[9] SingleClick.html
[10] Update.html
[11] Compilation.html
[12] httpd/ServerRoot.html
[13] httpd/ServerRoot.html





- `httpd` : The server itself
- `conf` : Configuration files
- `logs` : Access log and error log
- `support` : Support programs
- `cgi-bin` : Server scripts

**The logs directory should not be writable by the User[14] your server is running as.**

---

## Configuring HTTPd

The **NCSA HTTPd** Server is a versatile piece of software, the result of hours and hours of blood, sweat, and tears by the NCSA HTTPd Development team and Beta testers the world over. Its versatility does come at a price, however, as you the webmaster must configure the server to fit your needs. There are 3 types of configuration that can be done.

### Compile Time

The **NCSA HTTPd** server has several compile time flags. These are contained in the files `src/Makefile` and `src/config.h` of the distribution. These files are fairly well documented.
*See also: Compiling NCSA HTTPd*[15]

### Startup Configuration Files

There are 3 files the server parses at start up (or after receiving a SIGHUP signal). There is some General[16] information available on the configuration file format.

There are three configuration files which control Server Configuration[17], Resource Configuration[18], and Access Control[19]. You should look at and modify the files `conf/httpd.conf-dist`, `conf/srm.conf-dist` and `conf/access.conf-dist` which come with the distribution so that they are correct for your server.

### Run Time Configuration

The server also supports per directory configuration files, mostly for access control. The files in each directory of an accessed The name of this file is set with the AccessFileName[20] directive in the Resource Configuration[21] file. See the Access Configuration[22] documents for more information.

---

## Selecting Scripts

[14] httpd/User.html
[15] Compilation.html
[16] ConfigureInfo.html
[17] httpd/Overview.html
[18] srm/Overview.html
[19] access/Overview.html
[20] srm/AccessFileName.html
[21] srm/Overview.html
[22] access/Overview.html

**NCSA HTTPd 1.5** comes with many CGI scripts, both useful and informative. **NCSA HTTPd 1.5** does not support the former NCSA htbin scripts, and these scripts should be removed from the document tree. It is not necessary to install all of the scripts at your site (and we don't recommend it). Peruse the list[23] of CGI scripts, and move those you want into the `cgi-bin/` directory you defined in your Server Resource Configuration File[24].

---

## Starting NCSA HTTPd

Once you have completed the above steps, and you are using standalone[25] you can start the server by typing `httpd` at the command line. *Note: You must be root in order to use a port less than 1024.* You may need to use one of the command line options to override the compile time paths to the configuration files. The flags are:

```
Usage: httpd [-d directory] [-f file] [-v]
-d directory     : specify an alternate initial ServerRoot
-f file          : specify an alternate ServerConfigFile
-v               : version information (this screen)
```

If you are installing the server as root, you will probably want to automatically start HTTPd when the machine boots. This can be done through modifications to various /etc/rc* or /etc/init.d/* files, depending on your system.

If you are running the server from `inetd`, you need to edit the OS system files[26]. If you have problems starting your server, check out the FAQ[27]

---

If you have completed these steps, Congratulations! Your server is ready to go.

Before you run off to play with your new toy, *please* read this guide to referencing files on your server[28]. Particularly, if you are wondering how to set up your server's "home page", instructions are in there.

---

Return to the Overview[29]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[30] / 10−30−95*

[23] ScriptList.html
[24] srm/Overview.html
[25] httpd/ServerType.html
[26] StartInetd.html
[27] ../FAQ.html#Starting
[28] admin/AccessingFiles.html
[29] ../Overview.html
[30] ../FeedBack.html

# Downloading NCSA HTTPd

If you already have a copy of the current release of **NCSA HTTPd 1.5**, you may skip to the next step[1].

---

**NCSA HTTPd** is available as precompiled distributions[2] and binaries[3] for systems we have access to.

If you **already** have **NCSA HTTPd 1.4** running, you may update the HTTPd binary without the new support files. *Note: There are several new support files for the new authorization types, as well as examples of many of the other new features in the configuration files, so you might want to retrieve the support files anyways.* A step–by–step guide[4] on how to update the old server is available.

If your system is not on this list, or if you feel more comfortable doing so, you must compile a binary[5].

---

Back to Installation Overview[6]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[7] / 09–11–95*

[1] Configure.html
[2] PreCompiled.html
[3] PreExec.html
[4] Update.html
[5] Compilation.html
[6] Install.html
[7] ../FeedBack.html

# Precompiled Executables

If you **already** have NCSA HTTPd 1.5 running, you may update the HTTPd binary only without the support files. A step–by–step guide[1] on how to update the old server is available.

Select the item which matches your system to download a copy of the software.

You will get a .tar.Z file with the server binary, and other support directories.

- IRIX 4.0.5[2] – SGI Indigo
- IRIX 5.3[3] – SGI Indy
- SunOS 4.1.3 / Solaris 1.x[4] – SPARCserver 690MP
- SunOS 5.4 / Solaris 2.4 x86[5] – Pentium 90
- SunOS 5.4 / Solaris 2.4 SPARC[6] – SPARCstation 20
- SunOS 5.3 / Solaris 2.3 SPARC[7] – SPARCstation 20
- AIX 3.2.5[8] – IBM RS/6000 Model 550
- HP–UX 9.05[9] – HP 9000 model 715
- OSF/1 3.0[10] – Dec Alpha
- Ultrix 4.0[11] – Dec Mips 3100
- Linux 1.2.13, libc 5.0.9 ELF[12] – Pentium 120

If your system wasn't listed, you will need to compile[13] from the source.

---

Return to Installation Overview.[14]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[15] / 9–11–95*

---

1 Update.html
2 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_irix4.0.5.tar.Z
3 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_irix5.3.tar.Z
4 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_sunos4.1.3.tar.Z
5 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_solaris2.4_x86.tar.Z
6 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_solaris2.4_sparc.tar.Z
7 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_solaris2.3_sparc.tar.Z
8 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_aix3.2.5.tar.Z
9 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_hpux9.0.5.tar.Z
10 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_osf3.0.tar.Z
11 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_ultrix4.0.tar.Z
12 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_linux1.2.13_ELF.tar.Z
13 Compilation.html
14 Install.html
15 ../FeedBack.html

# NCSA HTTPd Precompiled Executables

Make sure you **already** have NCSA HTTPd 1.5 running before you download the executable here.

Select the item which matches your system to download a copy of compressed software *without* other support directories.

- IRIX 4.0.5[1] – SGI Indigo
- IRIX 5.3[2] – SGI Indy
- SunOS 4.1.3 / Solaris 1.x[3] – SPARCserver 690MP
- SunOS 5.4 / Solaris 2.4 x86[4] – Pentium 90
- SunOS 5.4 / Solaris 2.4 SPARC[5] – SPARCstation 20
- SunOS 5.3 / Solaris 2.3 SPARC[6] – SPARCstation 20
- AIX 3.2.5[7] – IBM RS/6000 Model 550
- HP–UX 9.05[8] – HP 9000 model 715
- OSF/1 3.0[9] – Dec Alpha
- Ultrix 4.0[10] – Dec Mips 3100
- Linux 1.2.13, libc 5.0.9 ELF[11] – Pentium 120

If your system wasn't listed, you will need to compile[12] from the source.

---

Return to Update page.[13]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[14] / 9–11–95*

[1] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_irix4.0.5.Z
[2] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_irix5.3.Z
[3] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_sunos4.1.3.Z
[4] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_solaris2.4_x86.Z
[5] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_solaris2.4_sparc.Z
[6] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_solaris2.3_sparc.Z
[7] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_aix3.2.5.Z
[8] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_hpux9.0.5.Z
[9] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_osf3.0.Z
[10] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_ultrix4.0.Z
[11] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/httpd_1.5/httpd_1.5a–export_linux1.2.13_ELF.Z
[12] Compilation.html
[13] Update.html
[14] ../FeedBack.html

# Compiling NCSA HTTPd

To compile NCSA HTTPd, just take the following easy steps:

- Retrieve the Source[1]
- Modify the Makefiles[2]
- Select options and modify src/config.h[3]
- Type make in each directory

## Retrieve the Source

First, download the source[4]. (*Note: Some browsers will automatically uncompress the source for you*) and use `tar` and `uncompress` to decode it.

e.g.: `uncompress -c httpd_1.5_source.tar.Z | tar -xf -`
or: `tar -xf httpd_1.5_source.tar`

## Modify the makefiles

If you don't have a supported configuration, or if you want to change the default configuration, you are going to have to edit the Makefile in each of the three directories (`support,src,cgi-src`). The supported configurations only require typing make from the top level directory and then reading the instructions, picking a system, and typing make again. v1.5 supports the following systems in this fashion:

- aix3 – IBM AIX 3.2.x
- aix4 – IBM AIX 4.1.x
- aux – Apple Unix A/UX
- bsdi – BSD/OS 2.x systems
- hp–cc – HP–UX 9.x machines using HP cc
- hp–gcc – HP–UX 9.x machines using gcc
- linux – Linux systems
- netbsd – NetBSD systems
- next – NeXTStep 3.x systems (and possibly earlier systems)
- osf1 – DEC Alpha based systems (OSF)
- sgi4 – Silicon Graphics IRIX 4.x
- sgi5 – Silicon Graphics IRIX 5.x
- solaris – Sun Solaris 2.x (aka SunOS 5.x)
- sunos – Sun SunOS 4.1.x (aka Solaris 1.x)
- svr4 – System 5 Release 4 systems
- ultrix – DEC Mips based systems (Ultrix)

The Makefiles in the support and cgi–src directories is fairly straight forward. The src directory is more fun. If your system is supported, uncomment the AUX_CFLAGS and EXTRA_LIBS line associated with it. There are defines available in src/Makefile for the following systems:

[1] #get_source
[2] #modify
[3] #config
[4] file://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/current/httpd_1.5a–export_source.tar.Z





| | | | |
|---|---|---|---|
| SunOS | Solaris2 | Irix | HP-UX |
| AIX3 | AIX4 | Ultrix | OSF/1 |
| Amdahl UTS 2.1 | Sequent | Linux | NetBSD |
| SCO ODT SCO SVR3.2 | SVR4 | NeXT | A/UX |
| HP/Apollo Domain/OS | AT&T SVR3.2 | QNX 4.22 | |

If your system is not listed, you have to modify `src/Makefile` and the Unknown system section of `src/portability.h`.

## Compile Time Configuration

We've tried to make most of the abilities of the server manageable at run time configuration, but this is not always possible or desirable. For that reason, some of the options are only available at compile time. All of these options can be dealt with by editting either `src/Makefile` or `src/config.h`. The ones in the Makefile are generally security related, and often have extra libraries and/or include file directories that are necessary for compilation. The options in `config.h` are usually no more than a `#define`, and are fairly well documented in `config.h`.

Later on in the installation, when you run HTTPd, you may see Night of the Living Dead on your machine, as defunct processes pile up and eventually bring it to a grinding halt. If you see this, you have the wrong setting of BSD.

Return to Installation Overview[5]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[6] / Last Modified 9–11–95*

[5] Install.html
[6] ../FeedBack.html

# Update NCSA HTTPd from 1.4.x to 1.5

In the NCSA HTTPd 1.5 release, our primary focus was on user requested features and security.

If you are running NCSA HTTPd 1.4.x, you may update the HTTPd binary only without downloading the whole package. However, to get the newest versions of the CGI scripts, you need to download the cgi−src directory from the distribution.

1. Download[1] the binary.
   Save the binary file in place of the old one. You don't need the old binary any more. Note: If you download the binary via a web browser, the browser has already uncompressed the file for you. If you download the binary via anonynous ftp, you should uncompress it using the command

   ```
   uncompress filename.Z
   ```

2. Terminate[2] the old server.
   Note: **Don't** use `kill -9 'cat pidfile'` to kill the server because `kill -9` will kill the parents but will not kill all the children. Use `kill 'cat pidfile'` instead.

3. Start[3] the new server.

---

Back to Installation Overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / 9−11−95*

[1] PreExec.html
[2] admin/KillIt.html#kill
[3] StartIt.html
[4] Install.html
[5] ../FeedBack.html

# NCSA HTTPd Configuration

These rules apply to all of HTTPd's configuration files.

- Case insensitive

  Except where pathnames are involved, these files are not case sensitive.

- Comment lines begin with #

  Lines which should be ignored begin with #, the hash sign. This must be the first character on the line. Comments must be on a line by themselves.

- One directive per line

  Each line of these files consists of:

  `Directive data [data2 ... datan]`

  `Directive` is a keyword HTTPd recognizes, followed by whitespace. `data` is specific to the directive. Any additional data entries should be separated by whitespace.

- Extra whitespace is ignored

  You can put extra spaces or tabs between `Directive` and `data`. To embed a space in `data` without separating it from any subsequent arguments use a \ character before the space.

---

Back to Configuration Overview[1]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / 9-11-95*

[1] Configure.html
[2] ../FeedBack.html

# Script Selection

In the HTTPd distribution, you will find a directory called `cgi-bin`. It contains the default server scripts.

You do not need to keep any of the scripts. They are mostly intended as examples and models from which to build your own scripts.

Here is a brief list of the scripts and what they do to help you decide whether or not you wish to enable them.

---

- archie[1]

  A very simple gateway to archie, using HTML <ISINDEX> tags.

- calendar[2]

  Another HTML <ISINDEX> example, this one an interface to the UNIX cal command.

- date[3]

  Prints the current date.

- donothing[4]

  Send back a HTTP status code of "204", which tells the browser to "stay put" on the current document.

- finger[5]

  A simple gateway to finger, using HTML <ISINDEX> tags.

- fortune

  Tell your fortune, mister?.

- imagemap

  Handles ISMAP queries. Read the setup[6] document for how to use it.

- jj[7]

  An simple example of online ordering.

[1] /cgi−bin/archie
[2] /cgi−bin/calendar
[3] /cgi−bin/date
[4] /cgi−bin/donothing
[5] /cgi−bin/finger
[6] /docs/tutorials/imagemapping.html
[7] /cgi−bin/jj





- mail[8]

  A mail script, written in PERL, to forward results from a FORM to pre−configured email addresses. Meant to be implemented once for an entire site, email addresses can be authorized based on domain.

- nph−error.pl

  An nph− script to help give your users a more informative message when encountering an error message on your system. Must be used with the ErrorDocument[9] directive in srm.conf.

- nph−test−cgi[10]

  An nph− script to echo back environment varibles.

- phf[11]

  An HTML form interface to the PH system.

- post−query

  A general purpose FORM response script, accepting METHOD="POST", and echoing back the name/value pairs submitted. Very useful for testing your FORMs.

- query

  A general purpose form response script, accepting METHOD="GET", and echoing back the name/value pairs submitted. Very useful for testing your FORMs.

- redirect[12]

  Script to bounce the browser to a new URL from a FORM selection.

- test−cgi[13]

  A script to echo back environment varibles.

- test−cgi.tcl

  A script written in TCL to echo back environment varibles.

- test−env[14]

  A simple shell script to echo back all environment varibles.

- uptime[15]

  Prints the server’s load average.

[8] /cgi−bin/mail
[9] http://hoohoo.ncsa.uiuc.edu/docs/setup/srm/ErrorDocument.html
[10] /cgi−bin/nph−test−cgi
[11] /cgi−bin/phf
[12] /cgi−bin/redirect
[13] /cgi−bin/test−cgi
[14] /cgi−bin/test−env
[15] /cgi−bin/uptime

- wais.pl[16]

  A very simple gateway to wais.

---

Return to the Installation Overview[17]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[18] / 11−11−95*

[16] /cgi−bin/wais.pl
[17] Install.html
[18] ../FeedBack.html

# Setting up under inetd

To set up inetd to recognize HTTPd and requests for it on a certain port, follow these instructions.

---

- Edit `/etc/services`

  Add a line which resembles the following to /etc/services:

  ```
  http            80/tcp
  ```

  You can replace 80 with whatever port number you want to serve from. This explanation of the port number[1] may help if you're confused.

- Edit `/etc/inetd.conf`

  Add a line which resembles the following to /etc/inetd.conf:

  ```
  http    stream  tcp     nowait  nobody /usr/local/etc/httpd/httpd httpd
  ```

  Replace `/usr/local/etc/httpd/httpd` with wherever you are keeping the server binary, and nobody with the user name you want requests fulfilled as.

- Restart inetd

  Find the `inetd` process using `ps`and `kill -HUP` it to restart it.

---

Back to Installation Overview[2]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / 9−11−95*

[1] ../setup/admin/AccessingFiles.html#Port
[2] Install.html
[3] ../FeedBack.html

# Types Configuration File format

Adventurous types may want to add types to their server for local use. We strongly recommend you use the AddTypes[1] directive instead.

---

Nonetheless, the format of the types configuration file is as follows.

Each line contains information for one http type. These types resemble MIME types. If you plan to add new ones, you should use subtypes beginning with x–, such as application/x–myprogram.

Lines beginning with # are comment lines, and suitably ignored.

Each line consists of:

`type/subtype ext1 ext2 ... extn`

`type/subtype` is the MIME–like type of the document.

`ext*` is any number of space–separated filename extensions which should be returned to the client if a file with the given extension is referenced.

---

Return to Configuration File Overview[2]

Return to TypesConfig directive information[3]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[4] / 9–11–95*

1 srm/AddType.html
2 Configure.html
3 httpd/TypesConfig.html
4 ../FeedBack.html

# SingleClick Downloader

There are many configuration directives to customize NCSA HTTPd to your liking. However, the seven directives[1] included below are the only ones you need to specify initially to get the server running. From there, you can go through are further customize the daemon to your needs.

---

## Initial: Operating System

First, you much choose which operating system version of the software you will download. If your particular system doesn't appear in the menu, then go with the default source, and you will have to compile it yourself.

If you don't know the platform you are on, go back to your command prompt, and type

```
machine%  uname -a
```

which will respond with a one−line description of your machine. Pay particular attention to the first word and the version number, and try to discern your operating system. For example, for one of the systems here at NCSA:

```
sangamon%  uname -a
SunOS sangamon.ncsa.uiuc.edu 5.3 Generic_101318-64 sun4 sparc
sangamon%
```

The result is "SunOS 5.3". Well, SunOS 5.3 is really Solaris 2.3, so you would select solaris2.3 from the menu.

Platform:

---

## Directive 1/7: Process type

This specifies exactly how your machine will run your HTTPd server. The preferred method is `standalone`, which will cause the server to be an independent daemon process. As `inetd`, the server binary will be reloaded into memory for every request, which can slow your server down[2].

ServerType[3]:

---

## Directive 2/7: Binding Port

[1] /docs/setup/ConfigureInfo.html#Directive
[2] /docs/setup/httpd/WhyNotInetd.html
[3] /docs/setup/httpd/ServerType.html





This specifies which port of your machine that the HTTPd daemon will bind to and listen for HTTP requests. The default port for HTTP requests is 80. If you wish to bind to a port below 1024, the daemon must be started as root for the proper permissions to do so. Otherwise, any port number above 1024 can be used by anyone. The maximum numbered port is 65535.

If your are the system administrator for the machine, and you are trying to set up a permanent HTTP daemon, then go with port 80. If you don't have root access to the machine, and just playing with HTTPd, use a port number that is easy to remember, like 8080.

Port[4]:

---

## Directive 3/7: Server user identity

This directive is only applicable if you are running the server as `standalone`, which you should. This is the user id the server will change to when answering requests and acting on files. As a result, all files to be served need to be readable by this user.

For minimal permissions, use the user "nobody". If you are system administrator, you might want to create a user "www" so you can more tightly control file permissions. If you are someone without root permissions, just use your own login name.

User[5]:

---

## Directive 4/7: Server group identity

Similarly, this directive is only applicable if you are running the server as `standalone`, which you should. This is the group id the server will change to when answering requests and acting on files. This allows for more flexibility in sharing files with the HTTPd daemon in a Unix filesystem environment.

For minimal permissions, use the group "nobody". If you are system administrator, you might want to create a group "www" so you can other users on your system have permission to edit the server's HTML files. If you are someone without root permissions, just use the name of your primary group, which will be the first one output by the **groups** command.

Group[6]:

---

## Directive 5/7: Server administrator email address

The `ServerAdmin` is the email address that the user should send an email message to when reporting a problem with the server. This address appears on the default

[4] /docs/setup/httpd/Port.html
[5] /docs/setup/httpd/User.html
[6] /docs/setup/httpd/Group.html

error messages displayed when there is a problem.

ServerAdmin[7]:

---

## Directive 6/7: Location of server directory

This directory specified by `ServerRoot` is where the server distribution will reside on your system. The recommended location is `/usr/local/etc/httpd`, but you need to be root to install this directory. If you don't have root permissions, choose a subdirectory in your home path, and insert the full path in the box. You can find out the path of your home directory with the **pwd** command.

ServerRoot[8]:

---

## Directive 7/7: Location of HTML files

The `DocumentRoot` directive tells the server where the HTML files to be served are. The default location is a subdirectory `htdocs` underneath the `ServerRoot`, so that it is isolated from the rest of your system files. However, feel free to make the DocumentRoot the home directory of the "www" user you just created, or a subdirectory in your home directory if you don't have root permissions.

DocumentRoot[9]:

---

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10]/ Last Modified 11−7−95*

[7] /docs/setup/httpd/ServerAdmin.html
[8] /docs/setup/httpd/ServerRoot.html
[9] /docs/setup/srm/DocumentRoot.html
[10] mailto:httpd@ncsa.uiuc.edu

# Constructing URLs to your server

Now that you have an http server, you'll want to reference it in your HTML documents and with your favorite Web browser.

Before you do anything, you should read this URL primer[1] to familiarize yourself with URLs. You will want to pay attention to the section referring to HTTP URLs.

---

## Definition

HTTP URLs have the basic form:

`http://servername:port/path`

### servername

Your server's full hostname. It can be the server's real name or a DNS alias.

### port

This is the port which your server is listening on. Specifying it in the URL is optional. If omitted, it is assumed to be 80.

If your ServerType[2] is `inetd`, your port number was set in `/etc/services`.

If your ServerType is `standalone`, the Port[3] directive set your port number.

### path

This is the path to the document. *This is not the absolute pathname of the document on your machine.*

The server translates `path` as follows:

1. It looks for any defined Alias[4] or ScriptAlias[5] virtual names at the beginning of `path`. If it finds one, it replaces the virtual name with the real name and processes the request.

2. It looks for a prefix of /~, and if UserDir[6] is not DISABLED, it will look in the user's public html subdirectory for the file.

3. It inserts DocumentRoot[7] at the beginning of `path` and processes the request.

---

## Setting up your Home Page

1 http://www.ncsa.uiuc.edu/demoweb/url–primer.html
2 ../httpd/ServerType.html
3 ../httpd/Port.html
4 ../srm/Alias.html
5 ../srm/ScriptAlias.html
6 ../srm/UserDir.html
7 ../srm/DocumentRoot.html





Some HTTP servers let you explicitly set up your home page (i.e. the page returned by the URL http://yourserver/ ).

To do this using NCSA HTTPd, create a DirectoryIndex[8] file in the DocumentRoot[9] directory.

Note that this index can be a symbolic link to another file.

---

## Examples

My Resource Configuration file[10] contains the following directives (among others):

```
DocumentRoot /u/Web
DirectoryIndex index.html
ScriptAlias /cgi-bin /usr/local/etc/httpd/cgi-bin
Alias /zftp /archive/ftp
```

An HTML document references `http://hoohoo.ncsa.uiuc.edu/docs/Overview.html`. The server finds no Alias or ScriptAlias virtual names in `path`, so it returns the file `/u/Web/docs/Overview.html`.

Someone references my home page as `http://hoohoo.ncsa.uiuc.edu/`. The server finds no virtual names, so it returns `/u/Web/index.html`.

Another HTML document references `http://hoohoo.ncsa.uiuc.edu/cgi-bin/uptime`. The server finds the ScriptAlias `/cgi-bin` at the beginning of `path`, and so executes the script `/usr/local/etc/httpd/cgi-bin/uptime`.

Another HTML document references `http://hoohoo.ncsa.uiuc.edu/zftp/README.txt`. The server finds the Alias `/zftp` at the beginning of `path`, and returns the file `/archive/ftp/README.txt`.

---

Return to administration overview[11]
Return to Installation overview[12]
Return to Imagemapping[13]
Return to Includes Tutorial[14]

8 ../srm/DirectoryIndex.html
9 ../srm/DocumentRoot.html
10 ../srm/Overview.html
11 Overview.html
12 ../Install.html
13 ../../tutorials/imagemapping.html
14 ../../tutorials/includes.html

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[15] / Last Modified 7−12−95*

[15] mailto:httpd@ncsa.uiuc.edu

# Configuration

Configuration means customizing HTTPd's configuration files to reflect your system and how HTTPd should act on your system.

Please read this general information[1] which applies to all of HTTPd's configuration files.

Also, after changing the main configuration files of your server, you need to restart[2] the server for the changes to take effect.

---

There are three main configuration files you must edit.

**Server Configuration**[3]
: Technical aspects of server operation.

**Resource Configuration**[4]
: Document and script locations and aliases.

**Access Control**[5]
: Control which clients access certain directories, and which features are available in different directories.

---

Back to Server Administration[6]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[7] / Last Modified 8−29−95*

---

[1] ../ConfigureInfo.html
[2] KillIt.html
[3] ../httpd/Overview.html
[4] ../srm/Overview.html
[5] ../access/Overview.html
[6] Overview.html
[7] ../../FeedBack.html

# Managing the Standalone Daemon

If you are using a ServerType[1] of `inetd`, you do not need to read these instructions.

---

If you make any changes to HTTPd's configuration files, you must restart the server. To do so, you will need to know where the server has logged its process id, given by the server configuration directive PidFile[2].

If `pidfile` is the setting of PidFile for your server, to restart the daemon execute the following on the command line:

```
kill -1 `cat pidfile`
```

or

```
kill -HUP `cat pidfile`
```

This will restart the daemon.

You should now check to see if the daemon restarted successfully. Check the last line of the server's error log, given by the server configuration directive ErrorLogFile[3], and make sure it says `HTTPd: successful restart`.

Alternatively, you may use the `ps` command to `grep` for the process id of the daemon.

---

If you ever need to terminate the server, use the following command:

```
kill `cat pidfile`
```

Node: **Don't** use ``kill -9 `cat pidfile` `` to kill the server because `kill -9` will kill the parents but will not kill all the children on some systems.

---

Return to the Administration Overview[4]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 8–29–95*

1 ../httpd/ServerType.html
2 ../httpd/PidFile.html
3 ../httpd/ErrorLog.html
4 Overview.html
5 ../../FeedBack.html

# Managing the Log Files

If you are using a ServerType[1] of `inetd`, you may remove or move the log files periodically using `mv` or `rm` with no effect on the server.

---

The standalone daemon opens the log files once when it starts up, and does not close them until killed. This is because the `fopen()` system call has a lot of overhead.

To rotate one of the log files, do the following:

1. Move the log file to a new name with the `mv` command

2. Restart the standalone daemon[2]

This will cause all new server processes to log to the new log file. You may then do what you wish with the old log file.

---

Return to the Administration Overview[3]
*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[4] / Last Modified 7−12−95*

[1] ../httpd/ServerType.html
[2] KillIt.html
[3] Overview.html
[4] mailto:httpd@ncsa.uiuc.edu

# Administrating the Server

Here is a reference to setting up, maintaining and utilizing the information and resources provided by your server.

---

How to:

**Changing your Configuration**[1]
General Information[2] – Server Configuration[3]
Resource Configuration[4] – Access Control[5]
**Access files on your server by constructing URLs to it.**[6]
**Restart or Terminate the standalone daemon**[7]
Archive or remove the access log file[8]
Set up the imagemap script[9]
Write CGI scripts to handle forms[10]
Use server side includes in your HTML documents[11]
Manage users for User Authentication[12]

---

Return to server overview[13]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[14] / Last Modified 7−12−95*

[1] Configuration.html
[2] ../ConfigureInfo.html
[3] ../httpd/Overview.html
[4] ../srm/Overview.html
[5] ../access/Overview.html
[6] AccessingFiles.html
[7] KillIt.html
[8] Logs.html
[9] ../../tutorials/imagemapping.html
[10] /cgi/
[11] ../../tutorials/includes.html
[12] UserManagement.html
[13] ../../Overview.html
[14] mailto:httpd@ncsa.uiuc.edu

# Managing Users

To use user authentication[1], you'll need to edit and manage user files[2] and group files[3].

## Using htpasswd to manage user files

To deal with user files, we provide a program in the support directory of the distribution called `htpasswd`. Usage:

```
htpasswd [-c] file user
```

The −c, if present, tells htpasswd to create a new passwd file of the specified name instead of editing an old one. `file` is the pathname of the user file you wish to edit. The `user` parameter is the name of the user you wish to add or edit.

If htpasswd finds the user you specified, it will ask you to change the user's password. Type the new password (it will ask twice). HTTPd will then update the file.

If htpasswd doesn't find the specified user, it will ask you to give the user an initial password.

## Group files

The format of the group file is as follows:

```
groupname: member1 member2 ...
```

Or, each line contains the name of a group, and a list of members separated by spaces.

Return to Administration Overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

1 ../../info/Security.html#userauth
2 ../access/AuthUserFile.html
3 ../access/AuthGroupFile.html
4 Overview.html
5 mailto:httpd@ncsa.uiuc.edu

# Advantages in new imagemap.c

## Advantage in new imagemap.c[1] are

- Users of imagemapping no longer need to bother the system administrator because they don't need to do anything with the server configuration files.
- Basically the imagemap.conf file is no longer needed.
- Your virtual URLs will work in this version. This kind of URL is a virtual pathname to a file on your server (i.e. a URL to your server[2] without the http://hostname part)
- Your #–ed URLs will also work.
- A new method called **point** has been added.
  - **Description**

    The point method implements the "closest point wins" scheme. So if different point methods are specified, the url is chosen according to the point closest to the click.
  - **Format**

    `point url coordinate`
- Backward Compatibility is provided so the old imagemaps will still work fine.

---

Return to the Tutorials on Httpd Image Mapping[3]

*NCSA HTTPd Development Team / http d@ncsa.uiuc.edu[4] / Last Modified 6–13–95*

[1] imagemap.txt
[2] /docs/setup/admin/AccessingFiles.html
[3] imagemapping.html
[4] mailto:httpd@ncsa.uiuc.edu

# Server Configuration File

*Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]*

When you unpacked HTTPd, there should have been a directory called `conf`. In there is a file called `httpd.conf-dist`. This is a template for your server configuration file. We suggest you use this file only for reference, and edit the file `conf/httpd.conf` as your server configuration file.

Here is a list of the directives this file recognizes. Following the link will give you usage information, default values, and examples.

- ServerType[9]
- Port[10]
- User[11]
- Group[12]
- ServerAdmin[13]
- ServerRoot[14]
- ServerName[15]
- StartServers[16]
- MaxServers[17]
- TimeOut[18]
- AccessConfig[19]
- ResourceConfig[20]
- TypesConfig[21]
- IdentityCheck[22]
- BindAddress[23]
- Host[24]
- VirtualHost[25]
- SRMOptions[26]
- Logging
    - ErrorLog[27]
    - TransferLog[28]

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ServerType.html
10 Port.html
11 User.html
12 Group.html
13 ServerAdmin.html
14 ServerRoot.html
15 ServerName.html
16 StartServer.html
17 MaxServer.html
18 TimeOut.html
19 AccessConfig.html
20 ResourceConfig.html
21 TypesConfig.html
22 IdentityCheck.html
23 BindAddress.html
24 Host.html
25 VirtualHost.html
26 SRMOptions.html
27 ErrorLog.html
28 TransferLog.html





    - AgentLog[29]
    - RefererLog[30]
    - RefererIgnore[31]
    - PidFile[32]
    - LogDirGroupWriteOk[33]
    - LogDirOtherWriteOk[34]
    - LogOptions[35]
- KeepAlive
    - KeepAlive[36]
    - KeepAliveTimeout[37]
    - MaxKeepAliveRequests[38]
- Security Related
    - AssumeDigestSupport[39]

---

Return to Configuration Instructions[40]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[41] / 10−02−95*

29 AgentLog.html
30 RefererLog.html
31 RefererIgnore.html
32 PidFile.html
33 LogDirGroupWriteOk.html
34 LogDirOtherWriteOk.html
35 LogOptions.html
36 KeepAlive.html
37 KeepAliveTimeout.html
38 MaxKeepAliveRequests.html
39 AssumeDigestSupport.html
40 ../Configure.html
41 ../../FeedBack.html

# AccessConfig Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The AccessConfig directive gives HTTPd the location of the global access configuration file[9].

---

## Syntax

`AccessConfig file`

`file` is the name of the global access configuration file. It is either a full pathname, or a partial pathname relative to ServerRoot[10].

Only one AccessConfig directive is allowed in the configuration file.

---

## Virtual Host

Access Information can be different across different hosts, but the configuration file is the same. For non–overlapping directory trees, you can merely specify different constraints for different directories. For overlapping trees, you must have different AccessFileNames[11] for each host and cannot use the global configuration.

---

## Default

If you do not specify an AccessConfig, HTTPd assumes:

`AccessConfig conf/access.conf`

Compile Option: `ACCESS_CONFIG_FILE`

---

## Examples

`AccessConfig conf/access–global`

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] ../access/Overview.html
[10] ServerRoot.html
[11] ../srm/AccessFileName.html





HTTPd looks for access configuration in the file conf/access–global in the ServerRoot[12] directory.

`AccessConfig /httpd/admin/access`

HTTPd looks for access configuration in the file `/httpd/admin/access`.

---

Return to Server Configuration File Overview[13]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[14] / Last Modified 7–12–95*

[12] ServerRoot.html
[13] Overview.html
[14] mailto:httpd@ncsa.uiuc.edu

# AgentLog Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The AgentLog directive tells HTTPd where to record client agent software. The UserAgent header, if present, is recorded in the format sent by the client. If LogOptions[9] is set to Combined, this directive does nothing.

This is for statistical purposes and the tracing of protocol violations. It should be included by all clients. The first white space delimited word must be the software product name, with an optional slash and version designator. Other products which form part of the user agent may be put as separate words.

## A User Agent Log Example

The timestamp is the same from the access_log.

```
[02/Oct/1995:12:00:28] SPRY_Mosaic/v7.36 (Windows 16-bit) SPRY_package/v4.00
[02/Oct/1995:12:00:28] Mozilla/1.1N (X11; I; SunOS 5.3 sun4c)
[02/Oct/1995:12:00:29] Mozilla/1.0N (Windows)
[02/Oct/1995:12:00:29] Mozilla/1.22 (Windows; I; 16bit)
[02/Oct/1995:12:00:29] NCSA Mosaic for the X Window System/2.4  libwww/2.12 mod
[02/Oct/1995:12:00:30] Mozilla/1.0N (Windows)
[02/Oct/1995:12:00:30] IBM WebExplorer DLL /v1.02  via proxy gateway  CERN-HTTP
[02/Oct/1995:12:00:31] NCSA Mosaic for the X Window System/2.4  libwww/2.12 mod
[02/Oct/1995:12:00:32] MacWeb/1.00ALPHA3  libwww/2.17
```

## Syntax

`AgentLog file`

`file` is the name of the file to which user agents will be logged. It is either a full pathname, or a partial pathname relative to ServerRoot[10].

## Virtual Host

This directive is valid in the virtual host section.

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 LogOptions.html
10 ServerRoot.html





## Default

If you do not specify an AgentLog, HTTPd assumes:

`AgentLog logs/agent_log`

Compile Option: `DEFAULT_AGENTLOG`

## Examples

`AgentLog logs/agents`

This logs user agents to the file logs/agents in the ServerRoot[11] directory.

`AgentLog /tmp/httpd-agents`

This logs user agents to the file `/tmp/httpd-agents`.

`AgentLog /dev/null`

This effectively turns off agent logging.

Return to Server Configuration File Overview[12]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[13] / 10−02−95*

11 ServerRoot.html
12 Overview.html
13 ../../FeedBack.html

# NCSA HTTPd Directive

*Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]*

## Purpose

## Syntax

## Virtual Host

## Default

Compile Option:

## Examples

Return to Configuration File Overview[9]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / 10−02−95*

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] Overview.html
[10] ../../FeedBack.html

# BindAddress Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

This is the second method for providing virtual host support. This allows for greater flexibility in configuration specific to each host, but is also more wasteful of resources. This method works by telling the server to only listen to a single address. This means that you need a separate copy of the server running for each virtual host address.

---

## Syntax

`BindAddress` *`servername`*

Where *servername* is a DNS name that maps to a single IP address, or an IP address.

---

## Virtual Host

This directive is incompatible with VirtualHost configuration. You cannot run both a BindAddress server and a VirtualHost server on the same machine.

---

## Default

The Default is `INADDR_ANY`, which is listen to all addresses.

---

## Examples

`BindAddress 127.0.0.1`

This would cause the server to only respond to connects to localhost.

`BindAddress *`

This would cause the server to respond to connects to any address. This is the default.

---

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/





Return to Configuration File Overview[9]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / 10–02–95*

[9] Overview.html
[10] ../../FeedBack.html

# ErrorLog Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The ErrorLog directive sets the file to which HTTPd will log errors and status messages it encounters. It currently logs the following:

- Startup Message, Restart Message, and Halt Message
- Debug Dumps (from USR1 signal)
- Clients which time out and abort
- Scripts which produce no output (malformed header from script)
- .htaccess[9] files which attempt to override things they do not have permission to
- Server bugs which produce a segmentation violation or bus error
- User Authentication[10] configuration problems
- File doesn't exist errors
- Resource Limit errors

---

## Syntax

`ErrorLog file`

`file` is the name of the file to which errors will be logged. It is either a full pathname, or a partial pathname relative to ServerRoot[11].

---

## Virtual Host

This directive is valid in the virtual host section.

---

## Default

If you do not specify an ErrorLog, HTTPd assumes:

`ErrorLog logs/error_log`

Compile Option: `DEFAULT_ERRORLOG`

---

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ../access/Overview.html
10 ../access/Overview.html#user_auth
11 ServerRoot.html





## Examples

`ErrorLog logs/errors`

This logs errors to the file logs/errors in the ServerRoot[12] directory.

`ErrorLog /tmp/httpd-errors`

This logs errors to the file `/tmp/httpd-errors`.

`ErrorLog /dev/null`

This effectively turns off error logging.

---

Return to Server Configuration File Overview[13]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[14] / 10−02−95*

12 ServerRoot.html
13 Overview.html
14 ../../FeedBack.html

# Group Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The Group directive sets which group id the server will answer requests as.

In order to use this directive, the standalone server must be initially run as `root`.

This directive does not mean that the original daemon process will run as the given group, it means that the children which answer requests run as the given group.

This directive is only applicable if you are using a ServerType[9] of `standalone`.

---

## Syntax

`Group id`

`id` is one of:

- A name

  Refer to the given group by name.

- # followed by a group number

  Refer to a group by its number.

Some systems require that this `id` must be a valid group found in `/etc/group`, or else a runtime server error will occur.

HPUX Systems cannot use nobody as a valid group.

---

## Virtual Host

Because of security limitations of the Unix `(TM)` security model, this directive can only be set once for the entire server.

---

## Default

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] ServerType.html





If you do not specify a Group, HTTPd assumes:

`Group #-1`

Compile Option: `DEFAULT_GROUP`

---

## Examples

`Group nogroup`

This will cause the server to run as group `nogroup`.

`Group #65536`

This will cause the server to run as group number 65536.

---

Return to Server Configuration File Overview[10]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / 10−02−95*

[10] Overview.html
[11] ../../FeedBack.html

# IdentityCheck Directive

*Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]*

## Purpose

This directive enables RFC931 compliant logging of the remote user name for sites which run identd or something similar. This information is logged in the access_log[9]. It should not be trusted in any way except for rudimentary usage tracking.

If you do not plan to use this information, keep this directive set to `off`. Your network administrator will thank you later.

---

## Syntax

`IdentityCheck setting`

`setting` is either `on` or `off`.

Only one IdentityCheck directive should appear in the configuration file.

---

## Virtual Host

This directive is not valid in the virtual host section because this information is determined before the server knows which host is being accessed.

---

## Default

If no IdentityCheck directive is present, httpd assumes:

`IdentityCheck off`

Compile Option: `DEFAULT_RFC931`

---

## Example

`IdentityCheck on`

Causes the server to check the identity of each person accessing it.

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 TransferLog.html





---

Return to Server Configuration File Overview[10]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / 10–02–95*

10 Overview.html
11 ../../FeedBack.html

# KeepAlive Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

In normal HTTP transactions, the browser opens up a separate connection to the server for everything the browser wants. So, if you have a page with 5 inline images, the browser will use separate connections to get the page and each inline image. With a browser which supports KeepAlive, the browsers requests the documents one at a time through the same connection. This eliminates the start up time and cost of each connection.

---

## Syntax

KeepAlive is either `On` or `Off`

---

## Virtual Host

This directive is not valid in VirtualHost configuration.

---

## Default

The default is Off.

Compile Option: `DEFAULT_ALLOW_KEEPALIVE`

---

## Examples

`KeepAlive On`

Turns on KeepAlive support.

---

Return to Configuration File Overview[9]

---

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 Overview.html





*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu*[10] */ 10−02−95*

10 ../../FeedBack.html

# KeepAliveTimeout Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

This directive specifies the maximum amount of time the server will wait between requests on an open KeepAlive connection.

## Syntax

`KeepAliveTimeout #num`

Where #num is number of seconds before the server times out and closes the connection.

## Virtual Host

This directive is not valid in VirtualHost configuration.

## Default

The default is 10 seconds.

Compile Option: `DEFAULT_KEEPALIVE_TIMEOUT`

## Examples

`KeepAliveTimeout 60`

This will allow the server to keep the connection open for a minute between requests.

Return to Configuration File Overview[9]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / 10−02−95*

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 Overview.html
10 ../../FeedBack.html

# NCSA HTTPd Directive

*Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]*

## Purpose

## Syntax

## Virtual Host

## Default

Compile Option:

## Examples

Return to Configuration File Overview[9]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / 10–02–95*

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 Overview.html
10 ../../FeedBack.html

# Directive

Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]

## Purpose

## Syntax

## Virtual Host

## Default

Compile Option:

## Examples

Return to Configuration File Overview[9]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / 10−02−95*

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 Overview.html
10 ../../FeedBack.html

# LogOptions Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The LogOptions directive allows you to switch between the Common Log File (CLF) format with extra information going to separate files, or a somewhat extended CLF format in which the extra information is logged at the end of the normal CLF information. It also allows you to change various other aspects of server logging.

Some Log Analysers won't accept the combined format, so you might want to check first.

## Example Combined Log Format

```
lothlorien.ncsa.uiuc.edu - - [19/Sep/1995:15:08:07 -0500] "GET / HTTP/1.0" 200
lothlorien.ncsa.uiuc.edu - - [19/Sep/1995:15:08:07 -0500] "GET /images/httpd.gi
lothlorien.ncsa.uiuc.edu - - [19/Sep/1995:15:08:07 -0500] "GET /images/mosaic-h
lothlorien.ncsa.uiuc.edu - - [19/Sep/1995:15:19:07 -0500] "GET /docs/Overview.h
lothlorien.ncsa.uiuc.edu - - [19/Sep/1995:15:19:07 -0500] "GET /images/icon.gif
```

Note the extra Referer field and the User agent field on the end

## Example of CLF with Servername

```
lothlorien.ncsa.uiuc.edu - - [06/Oct/1995:13:51:23 -0500] "GET /beta-1.5/howto/
lothlorien.ncsa.uiuc.edu - - [06/Oct/1995:13:51:34 -0500] "GET /beta-1.5/patche
lothlorien.ncsa.uiuc.edu - - [06/Oct/1995:13:51:35 -0500] "GET /icons/blank.xbm
lothlorien.ncsa.uiuc.edu - - [06/Oct/1995:13:51:35 -0500] "GET /icons/menu.gif
lothlorien.ncsa.uiuc.edu - - [06/Oct/1995:13:51:35 -0500] "GET /icons/text.gif
```

## Syntax

`LogOptions opt1 opt2`

Where `optn` is:

- `Combined`

  With this LogOption, the server doesn't open a RefererLog[9] or AgentLog[10]. The information normally logged to those files is now logged in the TransferLog[11].

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] RefererLog.html
[10] AgentLog.html
[11] TransferLog.html





- `Separate`

  With this LogOption, the logging behavior is reminiscent of NCSA HTTPd 1.4 with separate log files for RefererLog[12] and AgentLog[13].

- `Servername`

  With this LogOption, the name which the server used is logged in the TransferLog[14]. If you are using VirtualHost[15] support, this will be the name of VirtualHost.

- `Date`

  With this LogOption, the Date code section of the TransferLog[16] will be prepended to each line of the RefererLog[17] and AgentLog[18] to help with correlating the various files. This is only valid with Separate.

## Virtual Host

This directive is valid in virtual host configuration, and suggested if you intend to have a lot of virtual hosts as a way of reducing the number of file descriptors required.

## Default

The default is Separate.

## Examples

`LogOptions Separate`

This will have separate log files.

`LogOptions Combined ServerName`

This will make a single log file with as much information as possible.

Return to Configuration File Overview[19]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[20] / 10–02–95*

[12] RefererLog.html
[13] AgentLog.html
[14] TransferLog.html
[15] VirtualHost.html
[16] TransferLog.html
[17] RefererLog.html
[18] AgentLog.html
[19] Overview.html
[20] ../../FeedBack.html

# MaxKeepAliveRequests Directive

Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]

## Purpose

This directive specifies how many requests the client can make through the connection before the server closes it.

---

## Syntax

```
MaxKeepAliveRequests #num
```

Where #num is the number of requests the server will fulfill.
A #num of 0 implies no maximum, and the server will continue the KeepAlive transaction until the KeepAliveTimeout[9] or until the browser closes the connection.

---

## Virtual Host

This directive is not valid in VirtualHost configuration.

---

## Default

The default is 5.

Compile Option: `DEFAULT_KEEPALIVE_MAXREQUESTS`

---

## Examples

```
MaxKeepAliveRequests 0
```

The server will only close the connection if the timeout value is reached, or a non−KeepAlive capable request is fulfilled (any request for which the server cannot determine before sending the size of the request, including most CGI scripts and Server Side Includes).

---

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 KeepAliveTimeout.html





Return to Configuration File Overview[10]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / 10−02−95*

10 Overview.html
11 ../../FeedBack.html

# MaxServers Directive



## Purpose

The MaxServers directive sets the maximum number of children for the hunt group that NCSA HTTPd will launch to handle increased load from the number made at run time by StartServers[9].

For more information on the server hunt group, see the 1.4 Changes[10].

---

## Syntax

```
MaxServers number
```

---

## Default

If nothing is specified, the server will use 10.

Compile Option: `DEFAULT_MAX_DAEMON`

---

Return to Server Configuration File Overview[11]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[12] / Last Modified 7−12−95*

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] StartServer.html
[10] ../../Upgrade.html#1.4
[11] Overview.html
[12] mailto:httpd@ncsa.uiuc.edu

# PidFile Directive

*Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]*

## Purpose

The PidFile directive sets the file to which HTTPd records the daemon process id.

This directive is used only if ServerType[9] is `standalone`.

## Syntax

`PidFile file`

`file` is the name of the file to which the process id is logged. It is either a full pathname, or a partial pathname relative to ServerRoot[10].

Only one PidFile directive is allowed in the configuration file.

## Virtual Host

This directive is not valid in virtual host configuration.

## Default

If you do not specify a PidFile, HTTPd assumes:

`PidFile logs/httpd.pid`

Compile Option: `DEFAULT_PIDLOG`

## Examples

`PidFile logs/pid`

This logs the process id to the file logs/pid in the ServerRoot[11] directory.

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ServerType.html
10 ServerRoot.html
11 ServerRoot.html





`PidFile /tmp/httpd-pid`

This logs the process id to the file `/tmp/httpd-pid`.

`PidFile /dev/null`

The server does not log its process id.

Return to Server Configuration File Overview[12]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[13] / 10−02−95*

12 Overview.html
13 mailto:httpd@ncsa.uiuc.edu

# Port Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The Port directive sets what port HTTPd listens to for clients.

---

## Syntax

```
Port num
```

`num` is a number from 0 – 65536. Most ports below 1024, except port 80, are reserved by the system.

If you want to use a port below 1024 (such as 80, the standard HTTP port), and your ServerType[9] is `standalone`, you will need to run HTTPd as root when starting it up.

Only one Port directive is allowed in the configuration file.

---

## Virtual Host

The server can only listen to one port, so the Port directive is invalid in virtual host configuration.

---

## Default

If you do not specify a Port, HTTPd assumes:

```
Port 80
```

Compile Option: `DEFAULT_PORT`

---

## Examples

```
Port 8080
```

The server would listen to port 8080.

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ServerType.html





```
Port 84
```

The server would try to listen to port 84.

---

Return to Server Configuration File Overview[10]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / 10−02−95*

10 Overview.html
11 ../../FeedBack.html

# RefererIgnore Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The RefererIgnore directive tells HTTPd to ignore certain RefererLog[9] lines. If the RefererIgnore string is contained in the Referer header, then the line is not logged.

---

## Syntax

```
RefererIgnore string1 string2 ... stringn
```

`stringn` is the set of characters to look for in the Referer header. If this string is found, then the Referer is not logged. If there is no RefererIgnore directive, then all referer information is logged.

---

## Virtual Host

This directive is valid in virtual host configuration.

---

## Default

If you do not specify a RefererIgnore, HTTPd doesn't ignore logging any RefererLog[10] lines.

---

## Examples

```
RefererIgnore www.ncsa.uiuc.edu
```

This avoids logging referers from www.ncsa.uiuc.edu

```
RefererIgnore news
```

This avoids logging referers from news articles.

```
RefererIgnore www.cen www.ncsa
```

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 RefererLog.html
10 RefererLog.html





This avoids logging referers from either www.ncsa or www.cen

---

Return to Server Configuration File Overview[11]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[12] / 10−02−95*

11 Overview.html
12 ../../FeedBack.html

# RefererLog Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The RefererLog directive tells HTTPd where to record referers. The Referer header, if present, is recorded in the following format:

```
[Time Stamp] URI -> Document
```

Where `URI` is the Universal Resource Identifier for the document that references the server, and `Document` is the name of the file being refered to. This is a path relative to DocumentRoot[9]. It is also possible to ignore logging certain referers by using the RefererIgnore[10] directive.

This directive is non−functional under the Combined LogOption[11].

## A Referer Example Log

```
[02/Oct/1995:13:03:29] http://www.webcrawler.com/cgi-bin/WebQuery -> /images/ic
[02/Oct/1995:13:03:29] http://www.webcrawler.com/cgi-bin/WebQuery -> /images/ba
[02/Oct/1995:13:03:34] http://www.lysator.liu.se:7500/mud/faq/faq1.html -> /ftp
[02/Oct/1995:13:04:25] http://rhine.ece.utexas.edu/~shishir/ -> /docs/tutorials
[02/Oct/1995:13:04:30] http://humnet.humberc.on.ca/ -> /archie.html/
```

## Syntax

```
RefererLog file
```

`file` is the name of the file to which referers will be logged. It is either a full pathname, or a partial pathname relative to ServerRoot[12].

## Virtual Host

This directive is valid in virtual host configuration.

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ../srm/DocumentRoot.html
10 RefererIgnore.html
11 LogOptions.html
12 ServerRoot.html





## Default

If you do not specify a RefererLog, HTTPd assumes:

```
RefererLog logs/referer_log
```

Compile Option: `DEFAULT_REFERERLOG`

## Examples

```
RefererLog logs/referers
```

This logs referers to the file logs/referers in the ServerRoot[13] directory.

```
RefererLog /tmp/httpd-referers
```

This logs referers to the file `/tmp/httpd-referers`.

```
RefererLog /dev/null
```

This effectively turns off referer logging.

Return to Server Configuration File Overview[14]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[15] / Last Modified 7−12−95*

13 ServerRoot.html
14 Overview.html
15 mailto:httpd@ncsa.uiuc.edu

# ResourceConfig Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The ResourceConfig directive gives HTTPd the location of the resource configuration file[9].

## Syntax

`ResourceConfig file`

`file` is the name of the resource configuration file. It is either a full pathname, or a partial pathname relative to ServerRoot[10].

## Virtual Host

This directive is valid in virtual host configuration.

## Default

If you do not specify a ResourceConfig, HTTPd assumes:

`ResourceConfig conf/srm.conf`

Compile Option: `RESOURCE_CONFIG_FILE`

## Examples

`ResourceConfig conf/resources`

HTTPd looks for resource configuration in conf/resources in the ServerRoot[11] directory.

`ResourceConfig /httpd/admin/resources`

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ../srm/Overview.html
10 ServerRoot.html
11 ServerRoot.html





HTTPd looks for resource configuration in the file `/httpd/admin/resources`.

Return to Server Configuration File Overview[12]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[13] / 10−02−95*

12 Overview.html
13 ../../FeedBack.html

# NCSA HTTPd SRMOptions

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

This command is mostly for VirtualHost configuration. It allows you to specify Server Resource Management (SRM) configuration in the Server Configuration file. It works by calling the SRM parser on the file after the SRMOption line.

## Syntax

```
<SRMOptions>
Directive1 Config1
...
</SRMOptions>
```

Where *Directive1* is a SRM[9] directive. Any number of directives can be used in a SRMOptions section.

## Virtual Host

This directive is valid in both the VirtualHost section and the rest of the server configuration file. More than one section is also possible.

## Default

It works by default. There is no way to disable it.

Compile Option: *none*

## Examples

```
<SRMOptions>
DocumentRoot /www
Alias /icons/ /usr/local/etc/httpd/icons/
</SRMOptions>

<VirtualHost 127.0.0.1 required>
        ServerName localhost
        ServerAdmin yow@wow.net
```

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ../srm/Overview.html





```
        <SRMOptions>
                DocumentRoot /www/local
                Alias /icons/ /usr/local/etc/httpd/icons/
                ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-local/
                # Local can use internal imagemap
                AddType text/x-imagemap .map
                AddType application/x-httpd-cgi .cgi
        </SRMOptions>
</VirtualHost>
```

Return to Configuration File Overview[10]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / 11-06-95*

10 Overview.html
11 ../../FeedBack.html

# ServerAdmin Directive

Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]

## Purpose

The ServerAdmin directive gives the server your e−mail address. This is so that the server can give people your address to report error conditions.

---

## Syntax

`ServerAdmin address`

Where `address` is an e−mail address.

---

## Virtual Host

This directive is valid in virtual host configuration.

---

## Default

If you do not specify a ServerAdmin, HTTPd assumes nothing and prints no address for reporting errors.

Compile Option: `DEFAULT_ADMIN`

---

## Examples

`ServerAdmin www@widget.com`

Errors will have the address www@widget.com as the person to blame.

---

Return to Server Configuration File Overview[9]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / Last Modified*

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] Overview.html
[10] mailto:httpd@ncsa.uiuc.edu





*7−12−95*

# ServerName Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The ServerName directive[9] sets the hostname HTTPd should return when creating redirection URLs.

This directive should be used on systems where `gethostbyname` may not work on the local host, or where the hostname returned should be a DNS alias such as `www.widget.com`.

---

## Syntax

```
ServerName FQDN
```

Where `FQDN` is the full hostname (including domain name) to be returned as the server address.

---

## Virtual Host

This directive is valid in virtual host configuration.

---

## Default

If you do not specify a ServerName, HTTPd attempts to retrieve it through system calls.

---

## Example

```
ServerName www.widget.com
```

This would set the server's hostname as www.widget.com, as opposed to monster.widget.com as would be returned from the gethostname system call.

---

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ../ConfigureInfo.html#Directive





Return to Server Configuration File Overview[10]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / 10−02−95*

10 Overview.html
11 ../../FeedBack.html

# ServerRoot Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The ServerRoot directive sets the directory in which HTTPd lives.

Upon startup, HTTPd expects to find the Server Configuration File as conf/httpd.conf in the ServerRoot directory.

Other Server Configuration directives may use this directory to give relative paths for locations of files.

## Syntax

```
ServerRoot dir
```

Where `dir` is an absolute path of a directory on your server machine.

Only one ServerRoot directive is allowed in the server configuration file.

## Virtual Host

This directive is not available for virtual host configuration.

## Default

If you do not specify a ServerRoot, HTTPd assumes:

```
ServerRoot /usr/local/etc/httpd
```

Compile Option: `HTTPD_ROOT`

## Examples

```
ServerRoot /usr/local/httpd
```

This would set ServerRoot to the directory /usr/local/httpd.

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/





```
ServerRoot /web1/http
```

This would set ServerRoot to /web1/http.

Return to Server Configuration File Overview[9]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / 10−02−95*

9 Overview.html
10 ../../FeedBack.html

# ServerType Directive

*Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]*

## Purpose

The ServerType directive sets how the server is executed by the system.

## Syntax

`ServerType type`

`type` is one of:

- `inetd`

  To run the server from the system process `inetd`. The system then does all socket and child management.

- `standalone`

  To run the server as a daemon process. The server then does all socket and child management.

Only one ServerType directive is allowed in the configuration file.

## Virtual Host

This directive is set for the entire server and cannot be used in the VirtualHost section.

## Default

If you do not specify a ServerType, HTTPd assumes:

`ServerType standalone`

We recommend against running the server from inetd. Here's why[9].

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] WhyNotInetd.html





## Examples

`ServerType standalone`

`ServerType inetd`

Return to Server Configuration File Overview[10]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / 10–02–95*

[10] Overview.html
[11] ../../FeedBack.html

# StartServers Directive

Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]

## Purpose

The StartServers directive sets how many children NCSA HTTPd will launch at run time. NCSA HTTPd will continue launching children up to MaxServers[9]. For more information on the server hunt group, see the 1.4 Changes[10].

---

## Syntax

`StartServers number`

---

## Virtual Host

This directive is not used by virtual host configuration.

---

## Default

If no number is give, HTTPd will start 5 children.

Compile Option: `DEFAULT_START_DAEMON`

---

Return to Server Configuration File Overview[11]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[12] / 10−02−95*

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] MaxServer.html
[10] ../../Upgrade.html#1.4
[11] Overview.html
[12] ../../FeedBack.html

# TimeOut Directive

*Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]*

## Purpose

The TimeOut directive sets the amount of time the server will wait for a client to send its query once connected, or the maximum amount of time the server will spend waiting for a client to accept information.

---

## Syntax

`TimeOut time`

Where `time` is the amount of time in seconds the server should wait.

Only one TimeOut directive is allowed in the configuration file.

---

## Default

If you do not specify a TimeOut, HTTPd assumes:

`TimeOut 1200`

This is a timeout of twenty minutes.

Compile Option: `DEFAULT_TIMEOUT`

---

## Example

`TimeOut 600`

This sets the timeout to ten minutes. For large files served to slow network clients, you may need every second of it.

---

Return to Server Configuration File Overview[9]

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 Overview.html





*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[10] / 10−02−95*

10 ../../FeedBack.html

# TransferLog Directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

The TransferLog directive[9] tells HTTPd where to record client accesses.

The logfile format is as follows. Each line consists of:

```
host rfc931 authuser [DD/Mon/YYYY:hh:mm:ss] "request" ddd bbbb "opt_referer" "o
```

- host: Either the DNS name or the IP number of the remote client
- rfc931: Any information returned by identd for this person, – otherwise.
- authuser: If user sent a userid for authentication, the user name, – otherwise.
- DD: Day
- Mon: Month (calendar name)
- YYYY: Year
- hh: hour (24–hour format, the machine's timezone)
- mm: minutes
- ss: seconds
- request: The first line of the HTTP request as sent by the client.
- ddd: the status code returned by the server, – if not available.
- bbbb: the total number of bytes sent, *not including the HTTP/1.0 header*, – if not available
- opt_referer: the referer field if given and if LogOptions[10] is Combined
- opt_agent: the user agent field if given and if LogOptions[11] is Combined

You can determine the name of the file accessed through `request`.

## Syntax

`TransferLog file`

`file` is the name of the file to which transfers will be logged. It is either a full pathname, or a partial pathname relative to ServerRoot[12].

## Virtual Host

This directive is valid in virtual host configuration.

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ../ConfigureInfo.html#Directive
10 LogOptions.html
11 LogOptions.html
12 ServerRoot.html





## Default

If you do not specify a TransferLog, HTTPd assumes:

`TransferLog logs/access_log`

Compile Option: `DEFAULT_XFERLOG`

## Examples

`TransferLog logs/downloads`

This logs transfers to the file logs/downloads in the ServerRoot[13] directory.

`TransferLog /tmp/httpd-accesses`

This logs transfers to the file `/tmp/httpd-accesses`.

`TransferLog /dev/null`

This effectively turns off transfer logging.

Return to Server Configuration File Overview[14]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[15] / 10–02–95*

13 ServerRoot.html
14 Overview.html
15 ../../FeedBack.html

# TypesConfig Directive

*Top*[1] *– Docs*[2] *– General*[3] *– Server*[4] *– Resource*[5] *– Access*[6] *– FAQ*[7] *– Tutorials*[8]

## Purpose

The TypesConfig directive gives HTTPd the location of the typing configuration file.

This file is how HTTPd maps filename extensions to MIME types to return to HTTP/1.0 clients. You should not need to edit it. If you really want to, look at the file format[9].

## Syntax

`TypesConfig file`

`file` is the name of the types file. It is either a full pathname, or a partial pathname relative to ServerRoot[10].

Only one TypesConfig directive is allowed in the configuration file.

## Virtual Host

This directive is not valid for virtual host configuration.

## Default

If you do not specify an TypesConfig, HTTPd assumes:

`TypesConfig conf/mime.types`

Compile Option: `TYPES_CONFIG_FILE`

## Examples

`TypesConfig conf/mime-types`

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] ../TypesFormat.html
[10] ServerRoot.html





HTTPd looks for types configuration in the file conf/mime–types in the ServerRoot[11] directory.

`TypesConfig /httpd/admin/types-local`

HTTPd looks for types configuration in the file `/httpd/admin/types-local`.

Return to Server Configuration File Overview[12]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu*[13] */ 10–02–95*

[11] ServerRoot.html
[12] Overview.html
[13] ../../FeedBack.html

# NCSA HTTPd User Directive

*Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]*

## Purpose

The User directive sets which user id the server will answer requests as.

This directive is only applicable if you are using a ServerType[9] of `standalone`.

In order to use this directive, the standalone server process must be initially started as `root`.

This directive does not mean that the original daemon process will run as the given user, it means that the children which answer requests run as the given user.

## Syntax

`User id`

`id` is one of:

- A name

  Refer to the given user by name.

- # followed by a user number

  Refer to a user by their number.

Only one User directive is allowed in the configuration file.

## Virtual Host

Due to the limits of the Unix `(TM)` security architecture, the server cannot run as more than one user with the current server model while maintaining sufficient security on other fronts.

## Default

If you do not specify a User, HTTPd assumes:

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] ServerType.html





`User #-1`

Compile Option: `DEFAULT_USER`

## Examples

`User nobody`

This will cause the server to run as user nobody.

`User #-2`

This will cause the server to run as user number –2.

Return to Server Configuration File Overview[10]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / 10–02–95*

[10] Overview.html
[11] ../../FeedBack.html

# VirtualHost Directive

Top[1] – Docs[2] – General[3] – Server[4] – Resource[5] – Access[6] – FAQ[7] – Tutorials[8]

## Purpose

This directive allows you to customize the server to respond to different IP addresses in different ways. See the tutorial[9] for more information.

## Syntax

```
<VirtualHost servername errorlevel>
...
</VirtualHost>
```

Where :

- *servername*

  This is the DNS name or IP address of this virtual host. The DNS name should map to a single IP address.

- *errorlevel*

  This can be either `Required` or `Optional`. If set to Required, any misconfiguration or problem will cause the server start to fail. If this is set to Optional, the server will attempt to continue even if the configuration is invalid. This is an optional parameter, and if left off, the server will default to `Required`.

- ...

  This can be any number of Server[10] directives, including the SRMOptions[11] directive which allows Resource[12] directives in the server configuration file.

## Virtual Host

You cannot use VirtualHost directives within VirtualHost directives. You can use as many VirtualHost directives as you like, up to the limits of your servers memory and operating system.

[1] ../../../
[2] ../../
[3] ../ConfigureInfo.html
[4] ../httpd/Overview.html
[5] ../srm/Overview.html
[6] ../access/Overview.html
[7] ../../FAQ.html
[8] ../../tutorials/
[9] ../../tutorials/VirtualHost.html
[10] Overview.html
[11] SRMOptions.html
[12] ../srm/Overview.html





## Examples

```
<VirtualHost www.uiuc.net>
ServerName www.uiuc.net
ServerAdmin blong@uiuc.net
DocumentRoot /www
</VirtualHost>

<VirtualHost 127.0.0.1 Required>
        ServerName localhost
        ServerAdmin yow@wow.net
        <SRMOptions>
                DocumentRoot /www/local
                Alias /icons/ /usr/local/etc/httpd/icons/
                ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-local/
                # Local can use internal imagemap
                AddType text/x-imagemap .map
                AddType application/x-httpd-cgi .cgi
        </SRMOptions>
</VirtualHost>
```

Return to Configuration File Overview[13]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[14] / 10–02–95*

[13] Overview.html
[14] ../../FeedBack.html

# Why Not inetd?

Why should you run the server standalone as opposed to running it from inetd? If your server has a lot of CPU cycles to burn, then there is nothing wrong with running HTTPd from inetd. Most servers don't.

If you run the server from inetd, that means that whenever a request comes in, inetd must first fork a new process, then load the HTTPd binary. Once the HTTPd binary is loaded, HTTPd must then load and parse all of its configuration files (including httpd.conf, access.conf, and mime.types), which is quite a task to be done for each and every request that comes in.

Now, contrast this with running standalone. When HTTPd gets a request, it makes a copy of itself (which requires no loading of a binary since shared text pages are used), and the copy handles the request. The configuration files have already been loaded on startup, and so we don't reload them every time.

In fact, with NCSA HTTPd 1.4 and 1.5, it doesn't even have to make another copy of itself if there is an extra server already running. The parent will just pass off the request to the free child.

---

Return to ServerType instructions[1]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / 10−02−95*

[1] ServerType.html
[2] ../../FeedBack.html

# Setting up the Server Resource Map

When you unpacked HTTPd, there should have been a directory called `conf`. In there is a file called `srm.conf-dist`. This is a template for your resource configuration file. We suggest you use this file only for reference, and edit the file `conf/srm.conf` as your resource configuration file.

---

Here is a list of the directives[1] this file recognizes. Following the link will give you usage information, default values, and examples.

- DocumentRoot[2]
- UserDir[3]
- AccessFileName[4]
- Virtual URLs
  - Redirect[5]
  - Alias[6]
  - ScriptAlias[7]
- AddType[8]
- AddEncoding[9]
- DefaultType[10]
- Indexing Options
  - DirectoryIndex[11]
  - FancyIndexing[12]
  - DefaultIcon[13]
  - ReadmeName[14]
  - HeaderName[15]
  - AddDescription[16]
  - AddIcon[17]
  - AddIconByType[18]
  - AddIconByEncoding[19]
  - IndexIgnore[20]
  - IndexOptions[21]
- ErrorDocument[22]

---

1 ../ConfigureInfo.html#Directive
2 DocumentRoot.html
3 UserDir.html
4 AccessFileName.html
5 Redirect.html
6 Alias.html
7 ScriptAlias.html
8 AddType.html
9 AddEncoding.html
10 DefaultType.html
11 DirectoryIndex.html
12 FancyIndexing.html
13 DefaultIcon.html
14 ReadmeName.html
15 HeaderName.html
16 AddDescription.html
17 AddIcon.html
18 AddIconByType.html
19 AddIconByEncoding.html
20 IndexIgnore.html
21 IndexOptions.html
22 ErrorDocument.html





Return to Configuration Instructions[23]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[24] / Last Modified 7−12−95*

23 ../Configure.html
24 mailto:httpd@ncsa.uiuc.edu

# AccessFileName directive

## Purpose

When returning a document to a client, the server looks for access control files[1] in the document's directory as well as its parent directories. This directive[2] sets the name of the file HTTPd should look for to find access control files.

## Syntax

`AccessFileName file`

Where `file` is a file name.

Only one AccessFileName directive is allowed in the server configuration file.

## File

srm.conf

## Default

If you do not specify an AccessFileName, HTTPd assumes:

`AccessFileName .htaccess`

## Example

`AccessFileName .ncsa-acl`

This would set AccessFileName to .ncsa–acl.

Return to Resource Configuration File Overview[3]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[4] / Last Modified 7–12–95*

1 ../access/Overview.html
2 ../ConfigureInfo.html#Directive
3 Overview.html
4 mailto:httpd@ncsa.uiuc.edu

# NCSA HTTPd AddDescription directive

## Purpose

Tells HTTPd how to describe a file or a file type while generating a directory index.

## Syntax

`AddDescription "blah blah blah" fileid`

`fileid` is either a filename extension (like .html), a filename, a full real pathname to a file on your system, or a wildcard pattern[1] to match to filenames.

`blah blah blah` must be surrounded by quotes and is a short (preferably < 1 line) description of the file.

You may use as many AddDescription directives as you wish.

## File

srm.conf

## Default

There are no default descriptions.

## Example

`AddDescription "a great batch of nonsense" fargle.bargle`

A portion of HTTPd's index for fargle.bargle's directory would look something like this:

- foople.poople (8000 bytes)
- fargle.bargle : a great batch of nonsense(10000 bytes)

Return to Resource Configuration File Overview[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified*

[1] ../MatchExp.html
[2] Overview.html
[3] mailto:httpd@ncsa.uiuc.edu





*7-12-95*

# AddEncoding directive

## Purpose

Allows you to specify an encoding type for a document with a given filename extension.

In order to serve encoded documents to clients, the client must support the given encoding method, as well as the HTTP encoding extension.

## Syntax

`AddEncoding type extension`

`type` is the encoding type for the document.

`extension` is the filename extension to map to this encoding.

You may use as many AddEncoding directives as you wish.

## File

srm.conf

## Default

There are no default encodings.

## Example

`AddEncoding x-gzip gz`

This would cause any file ending in .gz to be marked as encoded using the x−gzip encoding method.

## Reference

See HTTP/1.0 Section 3.5 Content Codings[1]

[1] http://www.w3.org/hypertext/WWW/Protocols/HTTP1.0/draft−ietf−http−spec.html#Content−Codings





Return to Resource Configuration File Overview[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified 7−12−95*

[2] Overview.html
[3] mailto:httpd@ncsa.uiuc.edu

# AddIcon directive

## Purpose

Tells HTTPd what kind of an icon to show for a given filetype in a directory index.

You may wish to read the indexing tutorial[1] for more information.

## Syntax

`AddIcon icon name1 name2...`

`icon` is a virtual path[2] to an image file which should be shown for files which match the pattern of `names`. Alternatively, this can be a group of the format (alt,icon) where alt is the text tag given for an icon for non−graphical browsers, and icon is a virtual path.

`name` is either `^^DIRECTORY^^` for directories (or `**DIRECTORY**` for backward compatibility), `^^BLANKICON^^`, specifying the blank icon used to format the list properly, a file extension (like .html), a partial filename, a wildcard expression[3], or a complete physical pathname.

You may use as many AddIcon directives as you wish.

## File

srm.conf

## Default

There are no default icons.

## Example

`AddIcon /icons/image.xbm .gif .jpg .xbm`

When HTTPd is indexing and finds a file with the extension .gif, .jpg, or .xbm, it will reference `/icons/image.xbm` as an image to show next to the filename.

`AddIcon /icons/dir.xbm ^^DIRECTORY^^`

If a given directory entry is in fact a subdirectory, `/icons/dir.xbm` will be referenced as an image for that index entry.

1 ../../tutorials/index−tutorial.html
2 ../admin/AccessingFiles.html#path
3 ../MatchExp.html





`AddIcon (SND,/icons/sound.xbm) *.au`

This would reference `/icons/sound.xbm` as the image to show next to any sound file, with the textual ALT tag of SND for non−image clients.

Return to Resource Configuration File Overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

4 Overview.html
5 mailto:httpd@ncsa.uiuc.edu

# AddIconByEncoding directive

## Purpose

Tells HTTPd what kind of an icon to show for a given filetype in a directory index.

You may wish to read the indexing tutorial[1] for more information.

## Syntax

`AddIconByEncoding icon name1 name2...`

`icon` is a virtual path[2] to an image file which should be shown for files which match the pattern of `names`. Alternatively, this can be a group of the format (alt,icon) where alt is the text tag given for an icon for non−graphical browsers, and icon is a virtual path.

`name` is a wildcard expression[3] specifying the content−encoding for which to display this icon.

You may use as many AddIconByEncoding directives as you wish.

## File

srm.conf

## Default

There are no default icons.

## Example

`AddIconByEncoding /icons/compress.xbm x-compress`

When HTTPd is indexing, if it finds a file compressed with the UNIX compress command, it will display /icons/compress.xbm.

`AddIconByEncoding (CMP,/icons/compress.xbm) x-compress`

Same as above, but the ALT tag would be CMP for these items.

1 ../../tutorials/index−tutorial.html
2 ../admin/AccessingFiles.html#path
3 ../MatchExp.html





Return to Resource Configuration File Overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

4 Overview.html
5 mailto:httpd@ncsa.uiuc.edu

# AddIconByType directive

## Purpose

Tells HTTPd what kind of an icon to show for a given filetype in a directory index.

You may wish to read the indexing tutorial[1] for more information.

## Syntax

`AddIconByType icon type1 type2...`

`icon` is a virtual path[2] to an image file which should be shown for files which match the pattern of `types`. Alternatively, this can be a group of the format (alt,icon) where alt is a 3−letter text tag given for an icon for non−graphical browsers, and icon is a virtual path.

`name` is a wildcard expression[3] of the MIME types for which to add this icon.

You may use as many AddIconByType directives as you wish.

## File

srm.conf

## Default

There are no default icons.

## Example

`AddIconByType /icons/image.xbm image/*`

When HTTPd is indexing and finds a file which is an image, it will reference `/icons/image.xbm` as an image to show next to the filename.

`AddIconByType (SND,/icons/sound.xbm) audio/*`

This would reference `/icons/sound.xbm` as the image to show next to any sound file, with the textual ALT tag of SND for non−image clients.

1 ../../tutorials/index−tutorial.html
2 ../admin/AccessingFiles.html#path
3 ../MatchExp.html





Return to Resource Configuration File Overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

4 Overview.html
5 mailto:httpd@ncsa.uiuc.edu

# AddType directive

## Purpose

Allows you to add entries to the server's default typing information and cause an extension to be a certain type. These directives override any conflicting entries in the TypesConfig[1] file.

## Syntax

`AddType type/subtype extension`

`type/subtype` is the MIME–like type for the document.

`extension` is the filename extension to map to this type. This can either be a filename extension, a full pathname, or a file name.

You may use as many AddType directives as you wish.

## File

srm.conf

## Default

The default types are in the types configuration file.

## Example

`AddType text/plain doc`

This would cause any file ending in .doc to be served as type text/plain.

Return to Resource Configuration File Overview[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified 7–12–95*

1 ../httpd/TypesConfig.html
2 Overview.html
3 mailto:httpd@ncsa.uiuc.edu

# Alias directive

## Purpose

The Alias directive[1] creates a virtual document (or directory) on your server. Any accesses to it will be satisfied by a different file or directory.

## Syntax

`Alias virtual path`

`virtual` is the translated location[2] of the file/directory.

`path` is the full pathname of the file or directory which should be used to fulfill the request.

Several Alias directives may appear in the configuration file.

## File

srm.conf

## Example

`Alias /images /ftp/pub/images`

This would cause requests for `/images` to be satisfied from the directory `/ftp/pub/images`.

Thus, if someone requested `/images/foo.gif`, the server would return `/ftp/pub/images/foo.gif`.

Return to Resource Configuration File Overview[3]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[4] / Last Modified 7-12-95*

1 ../ConfigureInfo.html#Directive
2 ../admin/AccessingFiles.html
3 Overview.html
4 mailto:httpd@ncsa.uiuc.edu

# DefaultIcon directive

## Purpose

This directive specifies what icon should be shown in an automatically generated directory listing for a file which has no icon information.

## Syntax

`DefaultIcon location`

`location` is the virtual path[1] to the icon on your server.

Only one DefaultIcon directive should appear in the configuration file.

## File

srm.conf

## Default

If no DefaultIcon is present, HTTPd assumes nothing.

## Example

`DefaultIcon /icons/unknown.xbm`

This would cause a file with no icon to be given the icon /icons/unknown.xbm.

Return to Resource Configuration File Overview[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified 7−12−95*

1 ../admin/AccessingFiles.html#path
2 Overview.html
3 mailto:httpd@ncsa.uiuc.edu

# NCSA HTTPd DefaultType directive

## Purpose

If HTTPd can't type a file through normal means, it will type it as DefaultType.

## Syntax

`DefaultType type/subtype`

`type/subtype` is the MIME−like type.

Only one DefaultType directive should appear in the configuration file.

## File

srm.conf

## Default

If no DefaultType is present, HTTPd assumes:

`DefaultType text/html`

## Example

`DefaultType application/octet-stream`

This would cause a file with an unknown extension to be returned as type application/octet−stream.

Return to Resource Configuration File Overview[1]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / Last Modified 7−12−95*

[1] Overview.html
[2] mailto:httpd@ncsa.uiuc.edu

# DirectoryIndex directive

*Top*[1] – *Docs*[2] – *General*[3] – *Server*[4] – *Resource*[5] – *Access*[6] – *FAQ*[7] – *Tutorials*[8]

## Purpose

When a client requests a directory, HTTPd can return a pre−written index, or generate one from the filesystem. The DirectoryIndex directive[9] sets the file HTTPd should look for as a prewritten index to a given directory.

---

## Syntax

`DirectoryIndex file`

Where `file` is a file name.

Only one DirectoryIndex directive is allowed in the server configuration file.

---

## File

srm.conf

## Default

If you do not specify a DirectoryIndex, HTTPd assumes:

`DirectoryIndex index.html`

---

## Example

`DirectoryIndex .index.html`

This would set DirectoryIndex to .index.html. A request for `/dir` would cause the server to look for the file `DocumentRoot`[10]`/dir/.index.html`. If found, the server would send it back to the client. Otherwise, it would create and return an index from the filesystem.

---

1 ../../../
2 ../../
3 ../ConfigureInfo.html
4 ../httpd/Overview.html
5 ../srm/Overview.html
6 ../access/Overview.html
7 ../../FAQ.html
8 ../../tutorials/
9 ../ConfigureInfo.html#Directive
10 DocumentRoot.html





Return to Resource Configuration File Overview[11]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[12] / Last Modified 7−12−95*

11 Overview.html
12 mailto:httpd@ncsa.uiuc.edu

# DocumentRoot directive

## Purpose

The DocumentRoot directive[1] sets the directory from which HTTPd will serve files.

If you need to serve files outside this directory, you can use the Alias[2] directive, or create symbolic links.

This directive affects how you access files on your server[3].

## Syntax

`DocumentRoot dir`

Where `dir` is an absolute path of the directory you want documents to be served from.

Only one DocumentRoot directive is allowed in the server configuration file.

## File

srm.conf

## Default

If you do not specify a DocumentRoot, HTTPd assumes:

`DocumentRoot /usr/local/etc/httpd/htdocs`

## Example

`DocumentRoot /home/web`

This would set DocumentRoot to the directory /home/web.

Return to Resource Configuration File Overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified*

[1] ../ConfigureInfo.html#Directive
[2] Alias.html
[3] ../admin/AccessingFiles.html
[4] Overview.html
[5] mailto:httpd@ncsa.uiuc.edu





*7–12–95*

# ErrorDocument directive

## Purpose

The ErrorDocument directive[1] points the server to a file to send in place of the builtin error message.

### Syntax

```
ErrorDocument type filename
```

Where `type` is one of:

- 302 – REDIRECT
- 400 – BAD_REQUEST
- 401 – AUTH_REQUIRED
- 403 – FORBIDDEN
- 404 – NOT_FOUND
- 500 – SERVER_ERROR
- 501 – NOT_IMPLEMENTED

And `filename` is a CGI script[2] or text/html file with full path from document root. CGI scripts launched via these errors have 3 new environment variables, `REDIRECT_REQUEST`, `REDIRECT_URL` and `REDIRECT_STATUS`. They also take as input the error reason of the `form err_string=error_reason`. For an example script, see an example[3].

## File

srm.conf

## Default

If this directive is left out, the compiled error messages will be used.

## Examples

```
ErrorDocument 403 /cgi-bin/notallowed.cgi
ErrorDocument 404 /cgi-bin/nph-error.pl
ErrorDocument 500 /serverError.html
ErrorDocument 501 /error/notImplemented.html
```

[4]

For more information on Error scripts[5]

1 http://hoohoo.ncsa.uiuc.edu/docs/setup/ConfigureInfo.html#Directive</a
2 http://hoohoo.ncsa.uiuc.edu/cgi/overview.html
3 /cgi/ErrorCGI.html
4 /cgi/nph-error.pl
5 /cgi/ErrorCGI.html





Return to Resource Configuration File Overview[6]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[7] / Last Modified 7-12-95*

6 Overview.html
7 mailto:httpd@ncsa.uiuc.edu

# FancyIndexing directive

## Purpose

This directive specifies whether you want fancy directory indexing (with icons and file sizes) or standard directory indexing.

If your documents are being served from an AFS drive, you will want to seriously consider turning fancy indexing off.

## Syntax

`FancyIndexing setting`

`setting` is either `on` or `off`.

Only one FancyIndexing directive should appear in the configuration file.

## File

srm.conf

## Default

If no FancyIndexing directive is present, HTTPd assumes:

`FancyIndexing off`

## Example

`FancyIndexing on`

Turns fancy indexing on.

Return to Resource Configuration File Overview[1]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / Last Modified 7−12−95*

[1] Overview.html
[2] mailto:httpd@ncsa.uiuc.edu

# HeaderName directive

## Purpose

This directive specifies what filename HTTPd should look for when indexing a directory, in order to add a custom header. This can describe the contents of the directory.

## Syntax

`HeaderName name`

`name` is the name of the file HTTPd should look for when trying to find a description file. HTTPd will first look for `name.html`, and if found, will display the HTML inlined with its own index. If it finds `name`, it will include the file as plaintext.

Only one HeaderName directive should appear in the configuration file.

## File

srm.conf

## Default

If no HeaderName is present, HTTPd assumes nothing.

## Example

`HeaderName HEADER`

When generating an index for the directory `/foo`, HTTPd will look for `/foo/HEADER.html`, and will insert it at the top of the index if found. If not, it will then look for `/foo/HEADER` and insert it if found. If it finds nothing, it will include nothing.

Return to Resource Configuration File Overview[1]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / Last Modified 7−12−95*

[1] Overview.html
[2] mailto:httpd@ncsa.uiuc.edu

# IndexIgnore directive

## Purpose

Tells HTTPd which files to ignore when generating an index of a directory.

## Syntax

`IndexIgnore pat1 pat2...`

`pat` is a file extension or file name which should be ignored. When HTTPd is looking in a directory, it will try to match each of these strings to the right hand side of the entry's string, and if it matches it will ignore that entry in its directory index. If `pat` is a wildcard expression[1], HTTPd will match the filename against the given match expression.

## File

srm.conf

## Default

The only entry ignored by default is '.'.

## Example

`IndexIgnore README README.html .htaccess # ~`

HTTPd will ignore files named README, README.html, and .htaccess when indexing a directory. It will also ignore emacs autosave files and emacs backup files.

Return to Resource Configuration File Overview[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified 7−12−95*

1 ../MatchExp.html
2 Overview.html
3 mailto:httpd@ncsa.uiuc.edu

# IndexOptions directive

## Purpose

This directive specifies whether you want fancy directory indexing (with icons and file sizes) or standard directory indexing, and which options you want active for indexing.

If your documents are being served from an AFS drive, you will want to seriously consider turning fancy indexing off.

## Syntax

`IndexOptions opt1 opt2...`

`opt` is an option name:

- `FancyIndexing` turns FancyIndexing on
- `IconsAreLinks` makes the icons part of the anchor for the filename
- `ScanHTMLTitles` will cause HTTPd to fill in the description field of any unknown HTML document with its title (title must be within 256 bytes of the start of the file). You should NOT turn this option on unless your server has CPU time to spare.
- `SuppressLastModified` will cause HTTPd not to print the last date of modification in index listings.
- `SuppressSize` will cause HTTPd not to print the size of the files in index listings.
- `SuppressDescription` will cause HTTPd not to print descriptions for any files.

Only one IndexOptions directive should appear in the configuration file.

## File

srm.conf

## Default

If no IndexOptions directive is present, HTTPd assumes that none of the options should be on.

## Example





`IndexOptions FancyIndexing IconsAreLinks`

This will turn on fancy indexing with icons, and the icons will be part of the links to the files.

Return to Resource Configuration File Overview[1]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / Last Modified 7-12-95*

[1] Overview.html
[2] mailto:httpd@ncsa.uiuc.edu

# ReadmeName directive

## Purpose

This directive specifies what filename HTTPd should look for when indexing a directory, in order to add a paragraph of description to the end of the index it automatically generates. Generally these paragraphs are used to give a general overview of what's in a directory.

## Syntax

```
ReadmeName name
```

`name` is the name of the file HTTPd should look for when trying to find a description file. HTTPd will first look for `name.html`, and if found, will display the HTML inlined with its own index. If it finds `name`, it will include the file as plaintext.

Only one ReadmeName directive should appear in the configuration file.

## File

srm.conf

## Default

If no ReadmeName is present, HTTPd assumes nothing.

## Example

```
ReadmeName README
```

When generating an index for the directory `/foo`, HTTPd will look for `/foo/README.html`, and will insert it if found. It will then look for `/foo/README` and insert it if found. If it finds nothing, it will include nothing.

Return to Resource Configuration File Overview[1]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[2] / Last Modified 7−12−95*

[1] Overview.html
[2] mailto:httpd@ncsa.uiuc.edu

# Redirect directive

## Purpose

The Redirect directive[1] creates a virtual document on your server, and any accesses to it will be redirected to a new URL.

## Syntax

`Redirect virtual URL`

`virtual` is the translated location[2] which should trigger a redirect..

`URL` is the URL[3] of the new document.

Several Redirect directives may appear in the configuration file.

## File

srm.conf

## Example

```
Redirect /dir1 http://newserver.widget.com/dir1
```

This would cause requests for /dir1 to be redirected to the new location, `http://newserver.widget.com/dir1`.

Return to Resource Configuration File Overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

[1] ../ConfigureInfo.html#Directive
[2] ../admin/AccessingFiles.html
[3] http://www.ncsa.uiuc.edu/demoweb/url−primer.html
[4] Overview.html
[5] mailto:httpd@ncsa.uiuc.edu

# ScriptAlias directive

## Purpose

The ScriptAlias directive[1] creates a virtual directory on your server. Any accesses to that virtual directory will be satisfied by returning the output of a CGI server script[2] in that directory.

## Syntax

`ScriptAlias virtual path`

`virtual` is the translated location[3] of the script directory.

`path` is the full pathname of the directory which contains server scripts which fulfill the request.

Note: You should always place a trailing / after ScriptAlias directives which reference directories, to prevent similar entries from conflicting with each other.

Several ScriptAlias directives may appear in the configuration file.

## File

srm.conf

## Example

`ScriptAlias /cgi-bin/ /usr/local/etc/httpd/cgi-bin/`

This would cause requests such as `/cgi-bin/foo` to be satisfied by running the script `/usr/local/etc/httpd/cgi-bin/foo`.

Thus, if someone requested `/cgi-bin/a-script`, the server would run `/usr/local/etc/httpd/cgi-bin/a-script` and send its output to the client.

Return to Resource Configuration File Overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified*

1 ../ConfigureInfo.html#Directive
2 /cgi/
3 ../admin/AccessingFiles.html
4 Overview.html
5 mailto:httpd@ncsa.uiuc.edu





*7-12-95*

# UserDir directive

## Purpose

The UserDir directive[1] sets the real directory in a user's home directory to index upon receiving a request from a user supported directory[2].

## Syntax

`UserDir dir`

Where `dir` is a partial path relative to the users home directory (as given in `/etc/passwd`). The server takes the path it creates with this information, and looks for the user's document in the resulting full path.

The keyword `DISABLED` will disable the user supported directories feature.

Only one UserDir directive is allowed in the server configuration file.

## File

srm.conf

## Default

If you do not specify a UserDir, HTTPd assumes:

`UserDir public_html`

## Example

`UserDir web-docs`

This would set UserDir to web–docs. A request for `/~robm/foo.gif` would cause the server to retrieve `~robm/web-docs/foo.gif`.

Return to Resource Configuration File Overview[3]

1 ../ConfigureInfo.html#Directive
2 ../../info/UserDirectories.html
3 Overview.html

# Access Configuration

## Overview

When creating a document tree, NCSA HTTPd allows you to control certain aspects of the branches (directories) of that tree.

- Access

  Restrict access to a branch to allowed hosts or authenticated users.

- Server features

  Branches considered unsafe (such as users' home directories[1]) can be made more secure by disabling certain server functions in those directories.

## Related tutorials

There are many aspects of server administration regarding security and access control. Please read this tutorial[2] on setting up a secure server. Also, here is a quick overview[3] of setting up user authentication, which you may find helpful. If multiple users are using your server, this tutorial on user management[4] may also be helpfull.

## Methods

There are two methods for controlling access to directories.

- Global Access Configuration file

  A document in your server's `conf` directory, specified by the Server Configuration directive AccessConfig[5], controls access to any directory in your tree.

  HTTPd requires that you set up the Global ACF.

- Per−directory Access Configuration file

  Within your document tree, files with the name specified by the AccessFileName[6] directive in the Resource Configuration File[7] control access to the directory they are in as well as any subdirectories.

  Per−directory ACFs are optional. They can be restricted or completely forbidden by the Global ACF.

1 ../../info/UserDirectories.html
2 ../../tutorials/security.html
3 ../../tutorials/user.html
4 ../admin/UserManagement.html
5 ../httpd/AccessConfig.html
6 ../srm/AccessFileName.html
7 ../srm/Overview.html





## Directives

In addition to the general information[8] which applies to all NCSA HTTPd configuration files, the access control files support the notion of sectioning directives[9].

This is a list of the directives[10] and sectioning directives used when writing an ACF. Each directive specifies where it can be used, and gives examples of usage.

- Directory[11]
- Options[12]
- AllowOverride[13]
- AddType[14]
- DefaultType[15]
- AddEncoding[16]
- AddDescription[17]
- AddIcon[18]
- IndexIgnore[19]
- DefaultIcon[20]
- ReadmeName[21]
- AuthName[22]
- AuthType[23]
- AuthUserFile[24]
- AuthGroupFile[25]
- Limit[26]
  - order[27]
  - deny[28]
  - allow[29]
  - require[30]
  - satisfy[31]

## Example Configuration Files

For the security demo[32], I used this Global access configuration file[33].

8 ../ConfigureInfo.html
9 SectionDirectives.html
10 ../ConfigureInfo.html#Directive
11 Directory.html
12 Options.html
13 AllowOverride.html
14 AddType.html
15 DefaultType.html
16 AddEncoding.html
17 AddDescription.html
18 AddIcon.html
19 IndexIgnore.html
20 DefaultIcon.html
21 ReadmeName.html
22 AuthName.html
23 AuthType.html
24 AuthUserFile.html
25 AuthGroupFile.html
26 Limit.html
27 order.html
28 deny.html
29 allow.html
30 require.html
31 satisfy.html
32 ../../info/Security.html
33 access.conf.txt

The same effect could be achieved by taking everything within each Directory[34] section, and placing the directives in a ACF in each particular directory.

---

Return to Configuration Instructions[35]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[36] / Last Modified 7−12−95*

[34] Directory.html
[35] ../Configure.html
[36] mailto:httpd@ncsa.uiuc.edu

# AddDescription directive

## Purpose

The AddDescription directive[1] acts exactly as it does in the resource configuration file[2].

## Scope

This directive applies to only the per−directory ACFs.

## Syntax

See the description[3] from the resource configuration section.

## File

access.conf

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

1 ../ConfigureInfo.html#Directive
2 ../srm/Overview.html
3 ../srm/AddDescription.html
4 Overview.html#Directives
5 mailto:httpd@ncsa.uiuc.edu

# AddEncoding directive

## Purpose

The AddEncoding directive[1] acts exactly as it does in the resource configuration file[2].

## Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

## Syntax

See the description[3] from the resource configuration section.

## File

access.conf

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

[1] ../ConfigureInfo.html#Directive
[2] ../srm/Overview.html
[3] ../srm/AddEncoding.html
[4] Overview.html#Directives
[5] mailto:httpd@ncsa.uiuc.edu

# NCSA HTTPd AddIcon directive

## Purpose

The AddIcon directive[1] acts exactly as it does in the resource configuration file[2].

## Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

## Syntax

See the description[3] from the resource configuration section.

## File

access.conf

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

1 ../ConfigureInfo.html#Directive
2 ../srm/Overview.html
3 ../srm/AddIcon.html
4 Overview.html#Directives
5 mailto:httpd@ncsa.uiuc.edu

# AddType directive

## Purpose

The AddType directive[1] acts exactly as it does in the resource configuration file[2].

## Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

## Syntax

See the description[3] from the resource configuration section.

## File

access.conf

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

1 ../ConfigureInfo.html#Directive
2 ../srm/Overview.html
3 ../srm/AddType.html
4 Overview.html#Directives
5 mailto:httpd@ncsa.uiuc.edu

# AllowOverride directive

## Purpose

The AllowOverride directive[1] controls which access control directives can be overruled by a per−directory ACF.

The global ACF cannot be restricted by this directive.

## Scope

This directive may appear only in the global ACF.

## Syntax

`AllowOverride or1 or2 ... orn`

Each `or` is one of the following:

- None

  ACFs are not allowed in this directory.

- All

  ACFs are unrestricted in this directory.

- Options

  Allow use of the Options[2] directive.

- FileInfo

  Allow use of the AddType[3] and AddEncoding[4] directives.

- AuthConfig

  Allow use of these directives:
  - AuthName[5]
  - AuthType[6]
  - AuthUserFile[7]
  - AuthGroupFile[8]

[1] ../ConfigureInfo.html#Directive
[2] Options.html
[3] AddType.html
[4] AddEncoding.html
[5] AuthName.html
[6] AuthType.html
[7] AuthUserFile.html
[8] AuthGroupFile.html





- Limit

  Allow use of the Limit[9] sectioning directive.

- Redirect

  Allow use of the Redirect[10] directive.

## File

access.conf

## Default

If no AllowOverride directives are given for this directory or any of its parents, HTTPd assumes:

`AllowOverride All`

## Example

`AllowOverride Limit FileInfo`

ACFs in this directory are allowed to use `Limit`, `AddType`, and `AddEncoding`.

Return to access configuration overview[11]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[12] / Last Modified 7−12−95*

[9] Limit.html
[10] ../srm/Redirect.html
[11] Overview.html#Directives
[12] mailto:httpd@ncsa.uiuc.edu

# NCSA HTTPd AuthGroupFile

## Purpose

The AuthGroupFile directive[1] sets the file to use as a list of user groups for user authentication.

## Scope

This directive applies to both global and per−directory ACFs.

This directive must be accompanied by AuthName[2], AuthType[3], and AuthUserFile[4] directives in order for user authentication to work properly.

## Syntax

`AuthGroupFile path`

`path` is the absolute path of group file[5] to use in this directory.

## File

access.conf

## Default

There is no default.

## Example

`AuthGroupFile /usr/local/etc/httpd/conf/.htgroup`

Sets the authorization group file for this directory to `/usr/local/etc/httpd/conf/.htgroup`.

Return to access configuration overview[6]

1 ../ConfigureInfo.html#Directive
2 AuthName.html
3 AuthType.html
4 AuthUserFile.html
5 ../admin/UserManagement.html#group
6 Overview.html#Directives





*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[7] / Last Modified 7−12−95*

7 mailto:httpd@ncsa.uiuc.edu

# AuthName

## Purpose

The AuthName directive[1] sets the name of the authorization realm for this directory. This realm is a name given to users so they know which username and password to send.

## Scope

This directive applies to both global and per−directory ACFs.

This directive must be accompanied by AuthType[2], AuthUserFile[3], and AuthGroupFile[4] directives in order for user authentication to work properly.

## Syntax

```
AuthName name
```

Where name is a short name describing this authorization realm. Can contain spaces.

## File

access.conf

## Default

There is no default.

## Example

```
AuthName PhysicsCollab
```

Sets the authorization name of this directory to `PhysicsCollab`.

Return to access configuration overview[5]

1 ../ConfigureInfo.html#Directive
2 AuthType.html
3 AuthUserFile.html
4 AuthGroupFile.html
5 Overview.html#Directives





*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−12−95*[6]

6 mailto:httpd@ncsa.uiuc.edu

# AuthType

## Purpose

The AuthType directive[1] sets the type of authorization used in this directory.

## Scope

This directive applies to both global and per−directory ACFs.

This directive must be accompanied by AuthName[2], AuthUserFile[3], and AuthGroupFile[4] directives in order for user authentication to work properly.

## Syntax

`AuthType type`

`type` is the authentication type to use for this directory. Only `Basic` is currently implemented.

## File

access.conf

## Default

There is no default.

## Example

`AuthType Basic`

Sets the authorization type of this directory to `Basic`.

Return to access configuration overview[5]

1 ../ConfigureInfo.html#Directive
2 AuthName.html
3 AuthUserFile.html
4 AuthGroupFile.html
5 Overview.html#Directives





*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−12−95*[6]

6 mailto:httpd@ncsa.uiuc.edu

# AuthUserFile

## Purpose

The AuthUserFile directive[1] sets the file to use as a list of users and passwords for user authentication.

## Scope

This directive applies to both global and per−directory ACFs.

This directive must be accompanied by AuthName[2], AuthType[3], and AuthGroupFile[4] directives in order for user authentication to work properly.

## Syntax

`AuthUserFile path`

`path` is the absolute path of a user file created with the `htpasswd`[5] support program.

## File

access.conf

## Default

There is no default.

## Example

`AuthUserFile /usr/local/etc/httpd/conf/.htpasswd`

Sets the authorization user file for this directory to `/usr/local/etc/httpd/conf/.htpasswd`.

1 ../ConfigureInfo.html#Directive
2 AuthName.html
3 AuthType.html
4 AuthGroupFile.html
5 ../admin/UserManagement.html#htpasswd





Return to access configuration overview[6]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−12−95*[7]

6 Overview.html#Directives
7 mailto:httpd@ncsa.uiuc.edu

# DefaultIcon directive

## Purpose

The DefaultIcon directive[1] acts exactly as it does in the resource configuration file[2].

## Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

## Syntax

See the description[3] from the resource configuration section.

## File

access.conf

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−12−95*[5]

1 ../ConfigureInfo.html#Directive
2 ../srm/Overview.html
3 ../srm/DefaultIcon.html
4 Overview.html#Directives
5 mailto:httpd@ncsa.uiuc.edu

# DefaultType directive

## Purpose

The DefaultType directive[1] acts exactly as it does in the resource configuration file[2].

## Scope

This directive applies only to per−directory ACFs.

## Syntax

See the description[3] from the resource configuration section.

## File

access.conf

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−12−95*[5]

1 ../ConfigureInfo.html#Directive
2 ../srm/Overview.html
3 ../srm/DefaultType.html
4 Overview.html#Directives
5 mailto:httpd@ncsa.uiuc.edu

# Directory directive

## Purpose

Directory is a sectioning directive[1] which controls the directory to which access control directives apply.

## Scope

This directive applies only to the Global ACF. All directives[2] in the Global ACF must be contained in a Directory section.

## Syntax

Opening Directive:

```
<Directory dir>
```

`dir` is the absolute pathname of the directory you are protecting. If you are using NCSA HTTPd 1.2 or later, this can also be a wildcard expression[3] of a set of directories you wish to protect.

Closing directive:

```
</Directory>
```

## File

access.conf

## Example

```
<Directory /u/Web>
Options None
</Directory>
```

The directives contained within the Directory section above would only apply to the server directory /u/Web.

Return to access configuration overview[4]

1 SectionDirectives.html
2 ../ConfigureInfo.html#Directive
3 ../MatchExp.html
4 Overview.html#Directives





*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−12−95*[5]

5 mailto:httpd@ncsa.uiuc.edu

# IndexIgnore directive

## Purpose

The IndexIgnore directive[1] acts exactly as it does in the resource configuration file[2].

## Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

## Syntax

See the description[3] from the resource configuration section.

## File

access.conf

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

1 ../ConfigureInfo.html#Directive
2 ../srm/Overview.html
3 ../srm/IndexIgnore.html
4 Overview.html#Directives
5 mailto:httpd@ncsa.uiuc.edu

# Limit directive

## Purpose

Limit is a sectioning directive[1] which controls which clients can access a directory.

## Scope

This directive applies to both the global and per−directory ACFs.

## Syntax

Opening Directive:

```
<Limit meth1 meth2 ... methn>
```

Each `meth` is one of the following methods:

- `GET`

  Allows clients to retrieve documents and execute scripts.
- `PUT`

  Not Implemented.
- `POST`

  Not fully implemented. Currently, allows clients to use POST scripts.

Closing directive:

```
</Limit>
```

Only the following directives are allowed inside Limit sections:

- order[2]
- deny[3]
- allow[4]
- require[5]

## File

[1] SectionDirectives.html
[2] order.html
[3] deny.html
[4] allow.html
[5] require.html





access.conf

## Example

```
<Limit GET>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
require group sdg
</Limit>
```

The only clients allowed to use the `GET` method in this directory must be from ncsa.uiuc.edu, and authenticate to group sdg.

Return to access configuration overview[6]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[7] / Last Modified 7−12−95*

[6] Overview.html#Directives
[7] mailto:httpd@ncsa.uiuc.edu

# Options directive

## Purpose

The Options directive[1] controls which server features are available in a given directory.

## Scope

This directive applies to both the Global ACF as well as per–directory ACFs.

## Syntax

`Options opt1 opt2 ... optn`

Each `opt` is one of the following:

- None

  No features are enabled for in directory.

- All

  All features are enabled for in directory.

- FollowSymLinks

  The server will follow symbolic links in this directory.

- SymLinksIfOwnerMatch

  The server will only follow symbolic links for which the target file/directory is owned by the same user id as the link.

- ExecCGI

  Execution of CGI scripts[2] is allowed in this directory.

- Includes

  Server side include files[3] are enabled in this directory.

- Indexes

  The server allows users to request indexes[4] in this directory. Disabling this

1 ../ConfigureInfo.html#Directive
2 ../../tutorials/cgi.html
3 ../../tutorials/includes.html
4 ../../info/Indexes.html





  option disables ONLY the server–generated indexes. It does not stop the server from sending any precompiled index file it may find in there (the name of which depends on DirectoryIndex[5]).

- IncludesNoExec

  This enables server side includes[6] in the directory, but disables the exec[7] feature.

## File

access.conf

## Default

If no Options directives are given for this directory or any of its parents, HTTPd assumes:

`Options All`

## Example

`Options Indexes FollowSymLinks`

The server would allow users to index this directory and its subdirectories, and would allow symbolic links to be used within.

Return to access configuration overview[8]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[9] / Last Modified 7–12–95*

5 ../srm/DirectoryIndex.html
6 ../../tutorials/includes.html
7 ../../tutorials/includes.html#exec
8 Overview.html#Directives
9 mailto:httpd@ncsa.uiuc.edu

# ReadmeName directive

## Purpose

The ReadmeName directive[1] acts exactly as it does in the resource configuration file[2].

## Scope

This directive applies to both the Global ACF as well as per−directory ACFs.

## Syntax

See the description[3] from the resource configuration section.

## File

access.conf

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7−12−95*

1 ../ConfigureInfo.html#Directive
2 ../srm/Overview.html
3 ../srm/ReadmeName.html
4 Overview.html#Directives
5 mailto:httpd@ncsa.uiuc.edu

# Sectioning directives

## Definition

Some access control directives require that certain information apply to all directives in a given section. These directives are called sectioning directives.

The formatting of these directives is similar to HTML tags[1].

For each sectioning directive, there must be two components: an opening directive, and a closing directive.

For instance, for the sectioning directive `Foo`, with one argument `data`, the opening directive would be:

```
<Foo data>
```

The closing argument would be:

```
</Foo>
```

The information given in the opening directive will affect all other directives between the opening and closing sectioning directive.

## File

access.conf

## Example

The emphasized words are here for clarification; they would not appear in the actual configuration file.

```
<Directory /u/Web>                        The opening directive
require group physics
</Directory>                              The closing directive
```

`Directory` is the sectioning directive. Therefore, the information given by the opening directive, `/u/Web`, applies to the `require` directive within.

Return to access configuration overview[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified 7−12−95*

[1] http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html#MinimalHTML
[2] Overview.html#Directives
[3] mailto:httpd@ncsa.uiuc.edu

# allow directive

## Purpose

The allow directive affects which hosts can access a given directory with a given method.

## Scope

This directive is only available within Limit[1] sections.

## Syntax

`allow from host1 host2 ... hostn`

`host` is one of the following:

- A domain name

  A domain name, like .ncsa.uiuc.edu, which host names must end in to be allowed.
- A host name A full host name.
- A full IP address

  An IP address of a host.
- A partial IP address

  The first 1−3 bytes of an IP address, for subnet restriction.
- The keyword `all`

  This means that all hosts will be allowed.

## File

access.conf

## Default

No default applies.

[1] Limit.html





## Example

```
<Limit /u/Web>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
</Limit>
```

In the /u/Web directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu.

Return to access configuration overview[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified 7−12−95*

[2] Overview.html#Directives
[3] mailto:httpd@ncsa.uiuc.edu

# deny directive

## Purpose

The deny directive affects which hosts can access a given directory with a given method.

## Scope

This directive is only available within Limit[1] sections.

## Syntax

`deny from host1 host2 ... hostn`

`host` is one of the following:

- A domain name

  A domain name, like .ncsa.uiuc.edu, which host names must end in to be allowed.
- A host name A full host name.
- A full IP address

  An IP address of a host.
- A partial IP address

  The first 1−3 bytes of an IP address, for subnet restriction.
- The keyword `all`

  This means that all hosts will be denied.

## File

access.conf

## Default

No default applies.

[1] Limit.html





## Example

```
<Limit /u/Web>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
</Limit>
```

In the /u/Web directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu.

Return to access configuration overview[2]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / Last Modified 7−12−95*

[2] Overview.html#Directives
[3] mailto:httpd@ncsa.uiuc.edu

# order directive

## Purpose

The order directive affects the order in which deny[1] and allow[2] directives are evaluated within a Limit section.

## Scope

This directive is only available within Limit[3] sections.

## Syntax

```
order ord
```

`ord` is one of the following:

- `deny,allow`

  In this case, the deny directives are evaluated before the allow directives.

- `allow,deny`

  In this case, the allow directives are evaluated before the deny directives.

- `mutual-failure`

  This order is a bit unorthodox. With this order, you specify specific hosts which must be allowed or denied. Any host appearing on the allow list is allowed, and any list on the deny list is denied. Any appearing on neither is denied.

## File

access.conf

## Default

If no order is given, HTTPd assumes:

```
order deny,allow
```

[1] deny.html
[2] allow.html
[3] Limit.html





## Example

```
<Limit GET>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
</Limit>
```

In the /u/Web directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu.

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7-12-95*

[4] Overview.html#Directives
[5] mailto:httpd@ncsa.uiuc.edu

# require directive

## Purpose

The require directive affects which authenticated users can access a given directory with a given method.

## Scope

This directive is only available within Limit[1] sections.

## Syntax

`require entity en1 en2 ... enn`

`en` are entity names, separated by spaces.

`entity` is one of the following:

- `user`

  Only the named users[2] can access this directory with the given methods.
- `group`

  Only users in the named groups[3] can access this directory with the given methods.
- `valid-user`

  All of the users defined in the AuthUserFile[4] are allowed access upon providing a valid password.

## Default

No default applies.

## File

access.conf

[1] Limit.html
[2] ../admin/UserManagement.html
[3] ../admin/UserManagement.html#group
[4] AuthUserFile.html





## Example

```
<Limit GET PUT>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
require user ls
require group sdg
</Limit>
```

In this directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu. Now, it uses user authentication and only allows users who are named ls or are in the group sdg.

Return to access configuration overview[5]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[6] / Last Modified 7-12-95*

[5] Overview.html#Directives
[6] mailto:httpd@ncsa.uiuc.edu

# Satisfy directive

## Purpose

The satisfy directive allows you to specify how access is allowed if you use both allow[1] and require[2] directives.

## Scope

This directive is only available within Limit[3] sections.

## Syntax

`Satisfy all | any`

- `all`

  In order to gain access to a directory, the user must satisfy both the allow and require directives.
- `any`

  Users are only required to meet one of the specified allow or require directives.

## Default

The default is all.

## File

access.conf

## Example

```
<Limit GET PUT>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
require user ls
satisy any
</Limit>
```

[1] allow.html
[2] require.html
[3] Limit.html





In this directory, the server evaluates the deny directive first. So, everyone is denied. It then evaluates the allow directive, and decides to allow clients from .ncsa.uiuc.edu. With the satisfy any, if you are from .ncsa.uiuc.edu, access is granted. Otherwise, it uses user authentication and only allows users who are named ls.

Return to access configuration overview[4]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[5] / Last Modified 7-12-95*

[4] Overview.html#Directives
[5] mailto:httpd@ncsa.uiuc.edu

# NCSA HTTPd Tutorials

In an effort to make this documentation a bit more usable, we have written these tutorials on various aspects of server setup.

- **Directory Indexing**[1]
  FancyIndexing[2] – IndexOptions[3] – AddIcon[4] – AddIconByType[5] – DefaultIcon[6]
  AddDescription[7] – IndexIgnore[8] – ReadmeName[9] – HeaderName[10]
- **Server side Includes**[11]
  Options[12] – AddType[13] – CGI[14]
- **Security Related Tutorials**
  - Access control and user authentication[15]
    Setup[16] – AccessConfig[17] – AccessFileName[18]
    AuthName[19] – AuthType[20] – AuthUserFile[21] – AuthGroupFile[22]
  - Setting up a secure server[23]
  - Setting up a chroot server[24]
- **CGI Configuration**[25]
  - Imagemaps[26]
  - Using WAIS as a search engine for HTTP[27]
- Virtual Hosting[28]

---

Return to the documentation overview[29]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[30] / 9–11–95*

[1] index–tutorial.html
[2] ../setup/srm/FancyIndexing.html
[3] ../setup/srm/IndexOptions.html
[4] ../setup/srm/AddIcon.html
[5] ../setup/srm/AddIconByType.html
[6] ../setup/srm/DefaultIcon.html
[7] ../setup/srm/AddDescription.html
[8] ../setup/srm/IndexIgnore.html
[9] ../setup/srm/ReadmeName.html
[10] ../setup/srm/HeaderName.html
[11] includes.html
[12] ../setup/access/Options.html
[13] ../setup/srm/AddType.html
[14] /cgi/
[15] user.html
[16] ../setup/access/Overview.html
[17] ../setup/httpd/AccessConfig.html
[18] ../setup/srm/AccessFileName.html
[19] ../setup/access/AuthName.html
[20] ../setup/access/AuthType.html
[21] ../setup/access/AuthUserFile.html
[22] ../setup/access/AuthGroupFile.html
[23] security.html
[24] chroot.html
[25] cgi.html
[26] imagemapping.html
[27] wais.html
[28] VirtualHost.html
[29] ../Overview.html
[30] ../FeedBack.html

# Multihome / Virtualhost support

Multihome / Virtualhost support is a way in which a single machine can present two or more views of its documents based on the IP address that is accessed.

- What is Multihome / Virtualhost support?[1]
- Configuring NCSA HTTPd for Virtualhost Support[2]
- Configuring Solaris for Virtualhost Support[3]
- The VirtualHost Library[4]

---

## What is Multihome / Virtualhost support?

Every machine on the internet has an IP address. Most of them have at least one name which maps to that IP address via DNS. Some even have more than one name. Usually, this allows a provider to change which machine was providing a service by changing what this CNAME points to. Common CNAMES were mailhost, news, and gopher, until this little thing called the WWW came along. Now, www is probably one of the most common CNAMES. There's only one problem: CNAMES and DNS names in general, are meant to be transparent as far as the machine is concerned. So, although a machine could be both www.foo.org and www.bar.org, the machine has no way of knowing which name it was called by, since they map to the same IP address.

But, machines are also able to have more than one IP address. This was originally for gateways, machines which sit on two different subnets, and have a different IP address on each. Most machines now adays can have a different IP address per interface, and most machines have two IP addresses, 127.0.0.1 on the loopback interface, and another on the ethernet interface. Some platforms allow you to have more than one address on a single interface. They usually do this by providing Virtual Interfaces[5].

Multihome / Virtualhost / Virtual Interface support is the ability for NCSA HTTPd to recognize which interface (and thereby which IP address) the server was called from. You can then have www.foo.org and www.bar.org map to 2 different IP addresses, and the server can tell which one it is responding as.

Basically, the usual reason for this functionality is so that a company can provide a Web Presence (*Be On the Web*[6]) without most of the expense, by hiding behind a different Internet Service Provider. With it (as shown by the included configuration files) you can also provide a different service to users on the local machine (via the name localhost).

---

## Configuring NCSA HTTPd for Virtualhost Support

In the server configuration file (httpd.conf), after specifying the normal parameters

[1] #What
[2] #Config
[3] #Solaris
[4] #Library
[5] http://www.apache.org/docs/vif.info
[6] http://sdg.ncsa.uiuc.edu/~mag/work/specs/RFCs/rfc1775.txt





for your server, you start VirtualHost[7] section using the HTML–like syntax similar to the access configuration file (access.conf).
The syntax is:

```
<VirtualHost name>
Directive
Directive
</VirtualHost>
```

Where *name* is an IP address or hostname which corresponds to a single IP address. The directives that you can use are just about any that come from the other part of the server configuration file, including logfile names, the resource configuration file name, ServerAdmin, ServerName, and AnnotationServer. An example might be:

```
<VirtualHost 127.0.0.1>
ServerName localhost.foo.bar.com
ResourceConfig conf/localhost_srm.conf
TransferLog logs/localhost_access_log
</VirtualHost>
```

With this example, any accesses to the server by the name localhost will have the configuration options given override the other options. The server will parse the conf/localhost_srm.conf file and set its defaults accordingly if the server is called by the name localhost. All access to localhost will be logged in logs/localhost_access_log. All errors, referer, and user agent information will be logged to the default log files.

The resource configuration files for the virtual host are virtually identical to the regular configuration file, with some exceptions. Most of the fancy index'ing directives will work, but have unpredictable results. DocumentRoot, DirectoryIndex, AccessName, UserDir, DefaultType and DefaultIcon will override the default in the regular resource configuration file (srm.conf). In addition, ScriptAlias, Alias, Redirect, RedirectTemp and RedirectPermanent will only apply if you have them in the virtual host resource configuration file.

It should be able to handle any number of VirtualHosts up to the limits of the kernel and machine memory. Please place the virtual information after all the other information. If you don't, it might work, but the results can be unpredictably. Also be aware that most operating systems have a rather low number of open file descriptors allowed per process.

---

## Configuring Solaris for Virtualhost Support

From Solaris 2 FAQ[8]

**4.11) How can I have multiple addresses per interface?**

Solaris 2.x provides an undocumented feature in ifconfig that allows having more than one IP address per interfaces.

**Syntax:**

```
ifconfig IF:N ip-address up
```

where "IF" is an interface (e.g., le0) and N is a number between 1 and 255. Removing

[7] ../setup/httpd/VirtualHost.html
[8] ftp://ftp.fwi.uva.nl/pub/solaris/solaris2.html

the pseudo interface and associated address is done with "`ifconfig IF:N 0.0.0.0 down`".

---

## The Library

- RFC 1775 − Defining Being on the Web[9]
- Virtual Interface Information[10]
- Linux Virtual Howto[11]
- Solaris 2 FAQ[12]

---

Return to the tutorial index[13]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[14] / 10−20−95*

[9] http://sdg.ncsa.uiuc.edu/~mag/work/specs/RFCs/rfc1775.txt
[10] http://www.apache.org/docs/vif.info
[11] http://www.qosina.com/apache/virtual.html
[12] ftp://ftp.fwi.uva.nl/pub/solaris/solaris2.html
[13] index.html
[14] ../FeedBack.html

# CGI Configuration

CGI[1] scripts are a way for documents to be generated on the fly. You should first read this brief introduction[2] to CGI to learn what it is and why you would want to use it.

There are two main mechanisms to tell NCSA HTTPd where your scripts are. Each has its pluses and its minuses.

- ScriptAlias Directories[3]
- CGI Anywhere[4]

---

## ScriptAlias Directories

The first approach is based on the Server Resource Map[5] directive ScriptAlias[6]. With this directive, you specify to the server that you want to designate a directory (or directories) as script–only: that is, any time the server tries to retrieve a file from these directories it will execute the file instead of reading it.

The usual setup is to have the following line in srm.conf:

`ScriptAlias`[7] `/cgi-bin/ cgi-bin/`

This will make any request to the server which begins with `/cgi-bin/` be fulfilled by executing the corresponding program in `ServerRoot`[8]`/cgi-bin/`.

You may have more than one ScriptAlias directive in srm.conf to designate different directories as CGI.

The advantages of this setup are ease of administration, centralization, and slight speed advantage. Many system managers don't want things as dangerous as scripts anywhere in the filesystem. The disadvantage is that anyone wishing to create scripts must either have their own entry in srm.conf or must have write access to a ScriptAlias'd directory.

---

## CGI Anywhere

As of NCSA HTTPd 1.2, you can have the server execute CGI scripts anywhere. This is broken in different ways up until version 1.5, which should actually work completely, allowing full usage of CGI scripts in any location in the document tree, including if you use the AddType in .htaccess files.

NCSA HTTPd allows you to do this by by specifying a "magic" MIME type for files which tells the server to execute them instead of sending them. To accomplish this,

1 /cgi/
2 /cgi/intro.html
3 #ScriptAlias
4 #Anywhere
5 ../setup/srm/Overview.html
6 ../setup/srm/ScriptAlias.html
7 ../setup/srm/ScriptAlias.html
8 ../setup/httpd/ServerRoot.html





use the AddType[9] directive in either the Server Resource Map[10] or in a per–directory access control file[11].

For instance, to make all files ending in .cgi scripts, use the following directive:

`AddType`[12] `application/x-httpd-cgi .cgi`

Alternatively, you could add .sh and .pl after .cgi to allow automatic execution of shell scripts and PERL scripts. Note that you have to have `Options ExecCGI`[13] activated in the directory you create scripts.

The advantage of this setup is that scripts may be absolutely anywhere. The disadvantage is that scripts may be absolutely anywhere (especially places you don't want them to be like users' home directories).

---

Return to the tutorial index[14]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[15] / 9–27–95*

9 ../setup/srm/AddType.html
10 ../setup/srm/Overview.html
11 ../setup/access/Overview.html
12 ../setup/srm/AddType.html
13 ../setup/access/Options.html
14 index.html
15 ../FeedBack.html

# How to Use chroot for Increased Security

Every effort has been made on the part of the NCSA HTTPd Development Team to ensure a high level of security from break−ins through HTTPd. Occasionally, however, we miss one. For this reason, the server processes which handle outside connections are setuid to another user, such as *nobody*, who should have no permissions on the machine. This can be set using the User[1] directive.

For some, this might not be enough. There is something more that you can do, however. Most systems have available a command line `chroot` command. This command allows the system administrator (its usually restricted to the *root* user) to force a program to run under a subset of the file system, without allowing access from it to any other parts of the file system. This is usually how anonymous ftp is handled, and more information about running programs chroot can be gleaned from manpages and such which correspond to setting up anonymous ftp.

The general form of the command is:
`chroot` *`directory command`*

Where *directory* is the new root directory and *command* is the command to run under that directory.

For example:
`chroot /www httpd`

In this example, /www will become the new / directory, and anything not under /www will not be accessible.

There are some difficulties associated with this, however. Anything you want to server/run/etc. needs to be in /www. This also means that if HTTPd is linked with shared libraries, the shared libraries have to be available under /www. Any cgi scripts must be under /www, as well as the interpreters (sh,perl,etc) needed to run them. Any shared libraries for the cgi scripts need to be in /www. The document tree, the server root, the logfiles, the user directories (and a copy of /etc/passwd in /www/etc/passwd if you want to have /~username/ paths). You don't want to have the passwords in the /www/etc/passwd file, and you could fake a different user directory in it for /~username/ paths that is under the /www directory.

---

## This is a non−trivial thing to setup, and it is unlikely to be necessary.

We will provide only minimal support for setting up a chroot server, as we don't use this locally.

---

## Example

Using the following directory tree:

```
/www
```

[1] ../setup/httpd/User.html





```
/www/etc
/www/docs
/www/logs
/www/conf
/www/home
/www/home/blong
/www/home/httpd
/www/cgi-bin
/www/icons
/www/lib
/www/bin
```

Where `/www` will be the new root directory.

Into `/www/lib` you will need to copy at least the shared c library (`libc.so, libc.sa, libc.a, libc.sl`, or ..., depending on the system). Other libraries may be necessary depending on the system and what other binaries you want to run.

In `/www/bin`, you will need copies of sh, perl, tclsh, etc. If you have any gateways such as `archie, uptime, date, finger, ph,` copies of these programs will also have to be in in `/www/bin`.

`/www/docs` is your document root, but in the srm.conf file, it will be set to `/docs` (which is the directory after the chroot command).

`/www/etc` might need a copy of `/etc/passwd` and `/etc/group`. Make sure there are no passwords in this file (on most systems, this is the second field of the colon separated fields). Some systems require an entry for the uid of the user HTTPd will setuid to, so that entry would be the minimum for the passwd file (and group as well). If you intend to provide `/~username/` URLs, you must have a valid directory entry as well, which must be under `/www` but without the `/www` part. Here is a sample `/www/etc/passwd` file:

```
httpd:*:25:607:NCSA HTTPd Tech Support:/home/httpd:/bin/false
blong:*:524:1:Brandon Long:/home/blong:/bin/false
nobody:*:4294967294:4294967294::/:
```

`/www/conf` will contain all of the configuration files for HTTPd, and `/www/cgi-bin, /www/logs, /www/icons` contain what you would expect. All of the configuration parameters which point to an absolute directory must assume that `/www = /`

---

Return to tutorial overview[2]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[3] / 9−11−95*

[2] index.html
[3] ../FeedBack.html

# NCSA Imagemap Tutorial

Eternal gratitude to Kevin Hughes, *kevinh@pulua.hcc.hawaii.edu*, for his code to find intersections of points with circles and polygons.

This document is a step−by−step tutorial for designing and serving graphical maps of information resources with either the external imagemap CGI script or with the built in imagemap support in NCSA HTTPd 1.5. Through such a map, users can be provided with a graphical overview of any set of information resources; by clicking on different parts of the overview image, they can transparently access any of the information resources (possibly spread out all across the Internet).



*Note: Certain newer browsers support Client Side Imagemaps. For more information on Client Side Imagemaps, check out the Spyglass tutorial[7].*

---

## Requirements

This tutorial assumes use of NCSA HTTPd[8] version 1.0a5 or later. To use the internal imagemap support, you need NCSA HTTPd 1.5. Some other servers (e.g. Plexus, Apache, Netsite) can also serve imagemaps, in server−specific ways; see the specific server's docs for more information.

*Make sure you have a working NCSA HTTPd server installed and running*.

This tutorial also assumes use of a browser that supports inlined images, server side imagemapping and HTTP/1.0 URL Redirection.

---

## Compiling the Imagemap Script

If you downloaded or pre−compiled NCSA HTTPd server, or compiled your own, chances are the imagemap program was automatically compiled and added to your cgi−bin directory. If so, you can skip this step. If not, you can download the current imagemap.c from here[9].

Compile the imagemap script by first changing into your ServerRoot directory, and

[1] #Requirements
[2] #Compile
[3] #First
[4] #Internal
[5] #Example
[6] #Real
[7] http://www.spyglass.com/techspec/img_maps.html
[8] /docs/
[9] imagemap.txt





then into the cgi−src subdirectory. Put the new imagemap.c source in place of the old one (if you have the old version of imagemap.c) and then, type `make imagemap` and you should be all set.

---

## Your First Imagemap

In this section we walk through the steps needed to get an initial image map up and running.

*First: create an image.*

There are a number of image creation and editing programs that will work nicely −− the one I use is called *xpaint* (you can find it on *ftp.x.org* in /R5contrib[10]; here's the tar file[11]). It's for UNIX systems running an X interface.

*Second: create an imagemap map file.*

This file maps regions to URLs for the given image. For a list of tools that may help you create a map file, see Yahoo's Imagemap Directory[12]. For instance, there is a program called mapedit[13] that you could use.

A common scheme for an imagemap is a collection of points, polygons, rectangles and circles, each containing a short text description of some piece of information or some information server; interconnections are conveyed through lines or arcs. Try to keep the individual items in the map spaced out far enough so a user will clearly know what he or she is clicking on.

Lines beginning with # are comments. Every other non−blank line consists of the following:

```
method URL coord1 coord2 ... coordn
```

**method** is one of the following:

> **default** − For the default URL
>     Coordinates: none
> **circle** − For a circle
>     Coordinates: `center edgepoint`
> **poly** − For a polygon of at most 100 vertices
>     Coordinates: Each coordinate is a vertex.
> **rect** − For a rectangle
>     Coordinates: `upper-left lower-right`
> **point** − For closest to a point
>     Coordinate: `thePoint`

**URL** is one of the following:

> A Local URL
>     ex. `/docs/tutorials`
> A Full URL

[10] ftp://ftp.x.org/R5contrib/
[11] ftp://ftp.x.org/R5contrib/xpaint−2.1.1.tar.Z
[12] http://www.yahoo.com/Computers/Internet/World_Wide_Web/Programming/Imagemaps/
[13] http://sunsite.unc.edu/boutell/mapedit/mapedit.html

ex. `http://www.yahoo.com/`

**coord#**

Each coord entry is a coordinate, format x,y. The number depends on the **method**.

*Notes:*

- each method is evaluated in the order it is placed in the configuration file. If you have overlapping areas, such as a circle inside of a rectangle, you should place whichever one you want evaluated first before the other in the map file. In this case, we would put the circle before the rectangle.
- Also note that it does not make sense to use the default method with the point method because if even one point method is specified, anywhere you click will be considered close to the point and the URL specified by point will be serviced.
- If you will be serving access authentication[14]–protected documents through your imagemap, you MUST use fully qualified URLs, for example

  `http://your.server.com/path/to/protected/file.html`

  otherwise access will be denied.

Here is an example imagemap[15] linked from the Mosaic Demo page. Here is what a map file looks like:

```
default /X11/mosaic/public/none.html

rect http://cui_www.unige.ch/w3catalog    15,8    135,39
rect gopher://rs5.loc.gov/11/global       245,86  504,143
rect http://nearnet.gnn.com/GNN-ORA.html  117,122 175,158
```

The format is fairly straightforward. The first line specifies the default response (the file to be returned if the region of the image in which the user clicks doesn't correspond to anything).

Subsequent lines specify rectangles in the image that correspond to arbitrary URLs –– for the first of these lines, the rectangle specified by `15,8` (`x,y` of the upper–left corner, in pixels) and `135,39` (lower–right corner) corresponds to URL `http://cui_www.unige.ch/w3catalog`.

So, what you need to do is find the upper–left and lower–right corners of a rectangle for each information resource in your image map. A good tool to use on UNIX for this is *xv* (also on **ftp.x.org** in **/contrib**[16]) –– pop up the Info window and draw rectangles over the image with the middle mouse button.

It doesn't matter what you name your map file, **but it does matter where you put it!** Specifically, you cannot have your mapfile reside in the in the top–level of the

[14] /docs/info/Security.html
[15] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Demo/metamap.html
[16] ftp://ftp.x.org/contrib





DocumentRoot[17], because the imagemap CGI program will not see a "/" in the extra `PATH_INFO` that you are referencing. If this doesn't make sense to you, then just trust me: place your map file in a subdirectory.

*Third: referencing your imagemap in your HTML file.*

To reference your new map, you construct URLs[18] pointing to it.

For example, if your username is `newbie` and you have created a `sample.map` file in the directory called `path` in your `public_html` home directory, and used the image `sample.gif` for the map, the following line of HTML[19] will reference it:

```
<A HREF="http://www.school.edu/cgi-bin/imagemap/~newbie/foo/sample.map">
<IMG SRC="/~newbie/gifs/sample.gif" ISMAP>
</A>
```

The extra `PATH_INFO` after the `/cgi-bin/imagemap` tells the imagemap program where to find your map file. You'll note that it is possible to request the map file itself by simply requesting:

`http://www.school.edu/~newbie/foo/sample.map`

Or, using the internal imagemap support[20] of NCSA HTTPd 1.5:

```
<A HREF="http://www.school.edu/~newbie/foo/sample.map">
<IMG SRC="/~newbie/gifs/sample.gif" ISMAP>
</A>
```

*Fourth: Try it out!*

Load the HTML file, look at the inlined image, click somewhere, and see what happens.

---

## Internal Imagemap Support

With NCSA HTTPd 1.5, the server can now handle imagemap queries without launching the external imagemap script. This provides for a faster mechanism and decreased server load.

In order to enable the internal imagemap code, the server must be compiled with `IMAGEMAP_SUPPORT` defined. This is the default. Also, you need to add the magic mime type for the imagemap files in much the same way you would for CGI scripts[21]:

`AddType`[22] `text/x-imagemap .map`

[17] /docs/setup/srm/DocumentRoot.html
[18] /docs/setup/admin/AccessingFiles.html
[19] http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html
[20] #Internal
[21] cgi.html#Anywhere
[22] ../setup/srm/AddType.html

Then, you can reference the map file directly in your HTML instead of through the imagemap script, ie:

```
<A HREF="http://www.school.edu/~newbie/foo/sample.map">
<IMG SRC="/~newbie/gifs/sample.gif" ISMAP>
</A>
```

## A Complete Example

The fish demo[23] in another section of this manual used the following configuration files:

**Map Configuration File**

The map configuration file[24] used for this picture was rather lengthy. I used `xv` to get the coordinates.

**Reference the map in HTML document:**

```
<A HREF="/cgi-bin/imagemap/docs/info/fish.map"><IMG
SRC="fish33.gif" ISMAP> <A>
```

## Real-World Examples

Following are examples of distributed imagemaps on servers in the real world; they may or may not work at any point in time.

- Experimental Internet Resources Metamap[25] at NCSA.[26]
- University of California Museum of Paleontology.[27]
- National Institute of Standards and Technology.[28]
- Server map[29] at NCHPC Information Server.[30]

## For More Information

For more information, see the NCSA HTTPd documentation.[31]

Return to the tutorial index[32]

[23] /docs/info/Scripts.html#im
[24] fishmap.txt
[25] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Demo/metamap.html
[26] http://www.ncsa.uiuc.edu/General/NCSAHome.html
[27] http://ucmp1.berkeley.edu/
[28] http://www.nist.gov/welcome.html
[29] http://info.lcs.mit.edu/Info/structure.html
[30] http://info.lcs.mit.edu/
[31] /docs/
[32] /docs/tutorials/index.html





Return to administration overview[33]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[34] / 11-5-95*

[33] /docs/setup/admin/Overview.html
[34] ../FeedBack.html

# Server Side Includes (SSI)

NCSA HTTPd allows users to create documents which provide simple information to clients on the fly. Such information can include the current date, the file's last modification date, and the size or last modification of other files. In its more advanced usage, it can provide a powerful interface to CGI and /bin/sh programs.



---

## SSI Issues

Having the server parse documents is a double edged sword. It can be costly for heavily loaded servers to perform parsing of files while sending them. Further, it can be considered a security risk to have average users executing commands as the server's User[6]. If you disable the exec option, this danger is mitigated, but the performance issue remains. You should consider these items carefully before activating server–side includes on your server.

---

## SSI Setup

First, you should decide which directories you want to allow Includes in. Most likely this will not include users' home directories or directories you do not trust. You should then decide, of the directories you are allowing includes in, which directories are safe enough to use exec in.

For the directories in which you want to fully enable includes, you need to use the Options[7] directive to turn on the option `Includes`. Similarly for the directories you want crippled (no exec) includes, you should use the option `IncludesNOEXEC`. In any directory you want to disable includes, use the Options directive without either option.

Next, you need to tell the server what filename extension you are using for the parsed files. These files, while very similar to HTML, are *not* HTML and are thus not treated the same. Internally, the server uses the magic MIME type `text/x-server-parsed-html` to identify parsed documents. It will then perform a format conversion to change these files into HTML for the client. To tell the server which extension you want to use for parsed files, use the AddType[8] directive. For instance:

[1] #Issues
[2] #Setup
[3] #Convert
[4] #Format
[5] #Vars
[6] ../setup/httpd/User.html
[7] ../setup/access/Options.html
[8] ../setup/srm/AddType.html





```
AddType text/x-server-parsed-html .shtml
```
(AddType[9])

This makes any file ending with .shtml a parsed file. Alternatively, if you don't care about the performance hit of having all .html files parsed, you could use:

```
AddType text/x-server-parsed-html .html
```
(AddType[10])

This would make the server parse all .html files.

---

## Converting your old INC SRV documents to the SSI Format

You should use the program `inc2shtml` in the `support` subdirectory of the HTTPd distribution to translate your documents from HTTPd 1.1 and earlier to the new format. Usage is simple: `inc2shtml file.html > file.shtml`.

---

## The SSI Format

All directives to the server are formatted as SGML comments within the document. This is in case the document should ever find itself in the client's hands unparsed. Each directive has the following format:

```
<!--#command tag1="value1" tag2="value2" -->
```

Each command takes different arguments, most only accept one tag at a time. Here is a breakdown of the commands and their associated tags:

- `config`

  The config directive controls various aspects of the file parsing. There are two valid tags:

  - `errmsg` controls what message is sent back to the client if an error includes while parsing the document. When an error occurs, it is logged in the server's error log[11].
  - `timefmt` gives the server a new format to use when providing dates. This is a string compatible with the `strftime` library call under most versions of UNIX.
  - `sizefmt` determines the formatting to be used when displaying the size of a file. Valid choices are `bytes`, for a formatted byte count (formatted as 1,234,567), or `abbrev` for an abbreviated version displaying the number of kilobytes or megabytes the file occupies.

- `include`

  include will insert the text of a document into the parsed document. Any included file is subject to the usual access control. This command accepts two tags:

  - `virtual` gives a virtual path[12] to a document on the server. You must

[9] ../setup/srm/AddType.html
[10] ../setup/srm/AddType.html
[11] ../setup/httpd/ErrorLog.html
[12] ../setup/admin/AccessingFiles.html

access a normal file this way, you cannot access a CGI script in this fashion. You can, however, access another parsed document.

  - `file` gives a pathname relative to the current directory. ../ cannot be used in this pathname, nor can absolute paths be used. As above, you can send other parsed documents, but you cannot send CGI scripts.

- `echo` prints the value of one of the include variables (defined below). Any dates are printed subject to the currently configured `timefmt`. The only valid tag to this command is `var`, whose value is the name of the variable you wish to echo.
- `fsize` prints the size of the specified file. Valid tags are the same as with the `include` command. The resulting format of this command is subject to the `sizefmt` parameter to the `config` command.
- `flastmod` prints the last modification date of the specified file, subject to the formatting preference given by the `timefmt` parameter to `config`. Valid tags are the same as with the `include` command.
- `exec` executes a given shell command or CGI script. It must be activated to be used. Valid tags are:
  - `cmd` will execute the given string using /bin/sh. All of the variables defined below are defined, and can be used in the command.
  - `cgi` will execute the given virtual path to a CGI script and include its output. The server does not perform error checking to make sure your script didn't output horrible things like a GIF, so be careful. It will, however, interpret any URL Location: header and translate it into an HTML anchor.

---

## SSI Environment Variables

A number of variables are made available to parsed documents. In addition to the CGI variable set[13] the following variables are made available:

- `DOCUMENT_NAME`: The current filename.
- `DOCUMENT_URI`: The virtual path to this document (such as /docs/tutorials/foo.shtml).
- `QUERY_STRING_UNESCAPED`: The unescaped version of any search query the client sent, with all shell−special characters escaped with \.
- `DATE_LOCAL`: The current date, local time zone. Subject to the `timefmt` parameter to the `config` command.
- `DATE_GMT`: Same as DATE_LOCAL but in Greenwich mean time.
- `LAST_MODIFIED`: The last modification date of the current document. Subject to `timefmt` like the others.

[13] /cgi/env.html





Return to tutorial index[14]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[15] / 9−28−95*

[14] index.html
[15] ../FeedBack.html

# NCSA HTTPd Directory Indexing

NCSA HTTPd provides a directory indexing format which is similar to that which will be offered by the WWW Common Library. To set up this indexing, follow these steps.

- Activating Fancy Indexing[1]
- Choosing Icons[2]
- Adding Descriptions[3]
- Ignoring Items[4]
- Creating READMEs and HEADERs[5]

For an example of what these indexes look like, take a look at the demo[6].

---

## Activating Fancy Indexing

You first need to tell HTTPd to use the advanced indexing instead of the simple version. The simple version should be used if you prefer its simplicity, or if you are serving files off of a remote file server, for which the stat() call would be costly. You tell the server which you want to use with either the IndexOptions[7] directive, or the older FancyIndexing[8] directive. We recommend:

```
IndexOptions[9] FancyIndexing
```

---

## Choosing Icons

NCSA HTTPd comes with a number of icons in the `/icons` subdirectory which are used for directory indexing. As of 1.5, NCSA HTTPd comes with color versions of all of the icons. The `.xbm` files are black&white and the `.gif` files are color. The first thing you should do is make sure your Server Resource Map[10] has the following line in it:

```
Alias[11] /icons/ /usr/local/etc/httpd/icons/
```

You should replace `/usr/local/etc/httpd/` with whatever you set ServerRoot[12] to be.

Next, you need to tell the server what icons to provide for different types of files. You do this with the AddIcon[13] and AddIconByType[14] directives. We recommend something like the following setup:

[1] #Activation
[2] #Icons
[3] #Descriptions
[4] #Ignore
[5] #Readme
[6] ../info/star_trek/
[7] ../setup/srm/IndexOptions.html
[8] ../setup/srm/FancyIndexing.html
[9] ../setup/srm/IndexOptions.html
[10] ../setup/srm/Overview.html
[11] ../setup/srm/Alias.html
[12] ../setup/httpd/ServerRoot.html
[13] ../setup/srm/AddIcon.html
[14] ../setup/srm/AddIconByType.html





```
AddIconByType[15] (IMG,/icons/image.gif) image/*
AddIconByType[16] (SND,/icons/sound.gif) audio/*
AddIconByType[17] (TXT,/icons/text.gif) text/*
```

This covers the three main types of files. If you want to add your own icons, simply create the appropriately sized image, place it in `/icons`, and choose a 3–letter ALT identifier for the type.

HTTPd also requires three special icons, one for directories, one which is a blank icon the same size as the other icons, and one which specifies the parent directory of this index. To use the icons in the distribution, use the following lines in srm.conf:

```
AddIcon[18] /icons/menu.xbm ^^DIRECTORY^^
AddIcon[19] /icons/blank.xbm ^^BLANKICON^^
AddIcon[20] /icons/back.xbm ..
```

However, not all files fit one of these types. To provide a general icon for any unknown files, use the DefaultIcon[21] directive:

```
DefaultIcon[22] /icons/unknown.xbm
```

---

## Adding Descriptions

If you want to add descriptions to your files, use the AddDescription[23] directive. For instance, to add the description "My pictures" to `/www/my/images`, use the following line:

```
AddDescription[24] "My pictures" /www/my/images/*
```

If you want to have the titles of your HTML documents displayed for their descriptions, use the IndexOptions[25] directive to activate `ScanHTMLTitles`:

```
IndexOptions[26] FancyIndexing ScanHTMLTitles
```

**WARNING**: You should only use this option if your server has CPU time to spare!!! This is a *costly* operation!

---

## Ignoring Items

Generally, you don't want HTTPd sending references to certain files when it's creating indexes. Such files are emacs autosave and backup files, HTTPd's .htaccess files, and perhaps any file beginning with . (if you have a gopher or FTP server running in that directory as well). We recommend you ignore the following patterns:

[15] ../setup/srm/AddIconByType.html
[16] ../setup/srm/AddIconByType.html
[17] ../setup/srm/AddIconByType.html
[18] ../setup/srm/AddIcon.html
[19] ../setup/srm/AddIcon.html
[20] ../setup/srm/AddIcon.html
[21] ../setup/srm/DefaultIcon.html
[22] ../setup/srm/DefaultIcon.html
[23] ../setup/srm/AddDescription.html
[24] ../setup/srm/AddDescription.html
[25] ../setup/srm/IndexOptions.html
[26] ../setup/srm/IndexOptions.html

```
IndexIgnore[27] */.??* */README* */HEADER*
```

This tells HTTPd to ignore any file beginning with ., and any file starting with README or HEADER.

---

## Creating READMEs and HEADERs

When HTTPd is indexing a directory, it will look for two things and insert them into the index: A HEADER, and a README. Generally, the HEADER contains an HTML <H1> tag with a title for this index, and a brief description of what's in this directory. The README contains things you may want people to read about the items being served.

HTTPd will look for both plaintext and HTML versions of HEADERs or READMEs. If we add the following lines to srm.conf:

```
ReadmeName[28] README
HeaderName[29] HEADER
```

When HTTPd is indexing a directory, it will first look for HEADER.html. If it doesn't find that file, it will look for HEADER. If it finds neither, it will generate its own. If it finds one, it will insert it at the beginning of the index. Similarly, the server will look for README.html, then README to insert a trailer for the document.

---

Return to the tutorial index[30]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[31] / 9−27−95*

27 ../setup/srm/IndexIgnore.html
28 ../setup/srm/ReadmeName.html
29 ../setup/srm/HeaderName.html
30 index.html
31 ../FeedBack.html

# Making your setup more secure

When configuring the access control for your server, you will want to make sure you do not give any unauthorized access to anyone. Please follow these guidelines to ensure that your server is not compromised.

- A word of caution on DNS based access control and user authentication

  The access control by hostname and Basic user authentication facilities provided by HTTPd are relatively safe, but *not bulletproof*. The user authentication sends passwords across the network in plaintext, making them easily readable. The DNS based access control is only as safe as DNS, so you should keep that in mind when using it. Bottom line: If it absolutely positively cannot be seen by outside people, you probably should not use HTTPd to protect it.

- Disable Server−side includes wherever possible

  Whenever you can, use the Options[1] directive to disable server−side includes[2]. At the very least, you should disable the exec[3] feature. Note that because the default value of Options is All, you should include an Options directive in every Directory[4] clause in your global ACF and in every .htaccess file you write.

- Use AllowOverride[5] None wherever possible

  Use this directive to prevent any "untrusted" directories (such as users' home directories) from overriding your settings (and thus allowing their friends to execute xterms as nobody with a server−side include or other such horrors). You also gain a bonus in performance.

- Protect your users' home directories

  Protect your users' home directories with `Directory` directives. If your users all have their home directories in one physical location (such as `/home`), then this is easy:

```
<Directory /home>
AllowOverride None
Options Indexes
</Directory>
```

  If they are not all in one location such as /home, then you should use this wildcard pattern[6] to secure them (assuming your UserDir[7] is set to `public_html`):

```
<Directory /*/public_html*>
AllowOverride None
Options Indexes
</Directory>
```

[1] ../setup/access/Options.html
[2] includes.html
[3] includes.html#exec
[4] ../setup/access/Directory.html
[5] ../setup/access/AllowOverride.html
[6] ../setup/MatchExp.html
[7] ../setup/srm/UserDir.html





In addition, if you wish to give your users the ability to create symbolic links to things only they own, use the Option `SymLinksIfOwnerMatch`.

---

Return to tutorial overview[8]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[9] / 9−11−95*

[8] index.html
[9] ../FeedBack.html

# Mosaic User Authentication Tutorial

## Introduction

This tutorial surveys the current methods in NCSA Mosaic[1] and NCSA HTTPd[2] for restricting access to documents. The tutorial also walks through setup and use of these methods.

Mosaic 2.0 and NCSA HTTPd allow access restriction based on several criteria:

- Username/password–level access authorization.
- Rejection or acceptance of connections based on Internet address of client.
- A combination of the above two methods.

This tutorial is based heavily on work done by Ari Luotonen at CERN and Rob McCool at NCSA. In particular, Ari wrote the client–side code currently in Mosaic 2.0, and Rob wrote NCSA HTTPd 1.3.

---

## Tutorial Contents



---

## Getting Started

Before you can explore access authorization, you need to install[11] NCSA HTTPd 1.0a5 or later on a Unix machine under your control, or get write access to one or more directories in a filespace already being served by NCSA HTTPd. Other HTTP Servers also support access authentication, and some of the information presented here will help you understand how it works, but this tutorial is designed specifically for NCSA HTTPd.

NCSA HTTPd 1.5 has support for HTTP Basic Authentication (Basic), as well as the proposed Message Digest Authentication (MD5). Most, if not all, current browsers should support HTTP Basic Authentication, but not all browsers support MD5. Some

[1] http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/NCSAMosaicHome.html
[2] http://hoohoo.ncsa.uiuc.edu/docs/Overview.html
[3] #Introduction
[4] #Start
[5] #General
[6] #Secure
[7] #Basic
[8] #Multiple
[9] #Examples
[10] #More
[11] ../setup/Install.html





browsers that do include NCSA Mosaic/X 2.7[12] and Spyglass Mosaic[13]

---

## General Information

There are two levels at which authentication information can be passed to the server: the global access configuration file and the per–directory configuration files. This tutorial primarily covers per–directory configuration. See the NCSA HTTPd documentation[14] for information on global configuration.

Per–directory configuration means that users with write access to part of the filesystem that is being served (the Document Tree) can control access to their files as they wish. They need not have root access on the system or write access to the server's primary configuration files. Also, the per–directory configuration files are read and parsed by the server on each access, allowing run–time re–configuration. The global configuration files are only parsed on start–up or restart, which usually requires root authority. There is a speed penalty associated with using the per–directory configuration files, but that's the trade–off you have to take.

Access control for a given directory is controlled by a specific file in the directory with a filename as specified by the AccessFileName[15] directive. The default filename is `.htaccess`

---

## How Secure Is It?

In Basic HTTP Authentication, the password is passed over the network *not encrypted* but *not as plain text* –– it is "uuencoded." Anyone watching packet traffic on the network will not see the password in the clear, but the password will be easily decoded by anyone who happens to catch the right network packet.

So basically this method of authentication is roughly as safe as `telnet`–style username and password security –– if you trust your machine to be on the Internet, open to attempts to `telnet` in by anyone who wants to try, then you have no reason not to trust this method also.

In MD5 Message Digest Authentication, the password is not passed over the network at all. Instead, a series of numbers is generated based on the password and other information about the request, and these numbers are then hashed using MD5. The resulting "digest" is then sent over the network, and it is combined with other items on the server to test against the saved digest on the server. This method is more secure over the network, but it has a penalty. The comparison digest on the server must be stored in a fashion that it is retrievable. Basic Authentication stores the password using the one way `crypt()` function. When the password comes across, the server uudecodes it and then crypts it to check against the stored value. There is no way to get the password from the crypted value. In MD5, you need the information that is stored, so you can't use a one way hashing function to store it. This means that MD5 requires more rigorous security on the server machine. It is possible, but non–trivial, to implement this type of security under the Unix™ security model.

[12] http://www.ncsa.uiuc.edu/SDG/Software/XMosaic/index.html
[13] http://www.spyglass.com/products/browser.html
[14] http://hoohoo.ncsa.uiuc.edu/docs/Overview.html
[15] ../setup/srm/AccessFileName.html

## Basic ByPassword Authentication: Step By Step

This should help you set up protection on a directory via the Basic HTTP Authentication method. This method also uses the standard plaintext password file. If you have a large user base, NCSA HTTPd supports a DBM based password file for faster access.

So let's suppose you want to restrict files in a directory called `turkey` to username `pumpkin` and password `pie`. Here's what to do:

*Create a file called* `.htaccess` *in directory* `turkey` *that looks like this:*

```
AuthUserFile /otherdir/.htpasswd
AuthGroupFile /dev/null
AuthName ByPassword
AuthType Basic

<Limit GET>
require user pumpkin
</Limit>
```

Note that the password file will be in another directory (`/otherdir`).

Also note that in this case there is no group file, so we specify `/dev/null` (the standard Unix way to say "this file doesn't exist").

`AuthName` can be anything you want. The AuthName field gives the Realm name for which the protection is provided. This name is usually given when a browser prompts for a password, and is also usually used by a browser in correlation with the URL to save the password information you enter so that it can authenticate automatically on the next challenge. *Note: You should set this to something, otherwise it will default to ByPassword, which is both non−descriptive and too common.*

`AuthType` should be set to `Basic`, since we are using Basic HTTP Authentication. Other possibilities for NCSA HTTPd 1.5 are PEM, PGP, KerberosV4, KerberosV5, or Digest. These other types of authentication will be discussed later.

In this example, only the method GET is restricted using the `LIMIT` directive. To limit other methods (particularly in CGI directories), you can specify them separated by spaces in the `LIMIT` directive. For example:

```
<LIMIT GET POST PUT>
require user pumpkin
</LIMIT>
```

If you only use `GET` protection for a CGI script, you may be finding that the `REMOTE_USER` environment variable is not getting set when using `METHOD="POST"`, obviously because the directory isn't protected against `POST`.

*Create the password file* `/otherdir/.htpasswd`





The easiest way to do this is to use the `htpasswd` program distributed with NCSA HTTPd. Do this:

```
htpasswd -c /otherdir/.htpasswd pumpkin
```

Type the password −− `pie` −− twice as instructed.

Check the resulting file to get a warm feeling of self−satisfaction; it should look like this:

```
pumpkin:y1ia3tjWkhCK2
```

That's all. Now try to access a file in directory `turkey` −− your browser should demand a username and password, and not give you access to the file if you don't enter `pumpkin` and `pie`. If you are using a browser that doesn't handle authentication, you will not be able to access the document at all.

## Multiple Usernames/Passwords

If you want to give access to a directory to more than one username/password pair, follow the same steps as for a single username/password with the following additions:

*Add additional users to the directory's* `.htpasswd` *file.*

Use the `htpasswd` command without the `-c` flag to add additional users; e.g.:

```
htpasswd /otherdir/.htpasswd peanuts
htpasswd /otherdir/.htpasswd almonds
htpasswd /otherdir/.htpasswd walnuts
```

*Create a group file.*

Call it `/otherdir/.htgroup` and have it look something like this:

```
my-users: pumpkin peanuts almonds walnuts
```

`...` where `pumpkin`, `peanuts`, `almonds`, and `walnuts` are the usernames.

*Then modify the* `.htaccess` *file in the directory to look like this:*

```
AuthUserFile /otherdir/.htpasswd
AuthGroupFile /otherdir/.htgroup
AuthName ByPassword
AuthType Basic

<Limit GET>
require group my-users
</Limit>
```

Note that `AuthGroupFile` now points to your group file and that group `my-users` (rather than individual user `pumpkin`) is now required for access.

That's it. Now any user in group `my-users` can use his/her individual username and password to gain access to directory `turkey`.

---

## Prepared Examples

Following are several examples of the range of access authorization capabilities available through Mosaic and NCSA HTTPd. The examples are served from a system at NCSA.

*Simple protection by password.*

> This document[16] is accessible only to user `fido` with password `bones`.
>
> **Important Note**: There is *no* correspondence between usernames and passwords on specific Unix systems (e.g. in an `/etc/passwd` file) and usernames and passwords in the authentication schemes we're discussing for use in the Web. As illustrated in the examples, Web-based authentication uses similar but *wholly distinct* password files; a user need never have an actual account on a given Unix system in order to be validated for access to files being served from that system and protected with HTTP-based authentication.

*Protection by password; multiple users allowed.*

> This document[17] is accessible to user `rover` with password `bacon` and user `jumpy` with password `kibbles`.

*Protection by network domain.*

> This document[18] is only accessible to clients running on machines inside domain `ncsa.uiuc.edu`.
>
> *Note for non-NCSA readers:* The `.htaccess` file used in this case is as follows:

[16] /examples/auth/by-password/access-me.html
[17] /examples/auth/by-password-multiple-users/access-me.html
[18] /examples/auth/by-domain/access-me.html





```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName ExampleAllowFromNCSA
AuthType Basic

<Limit GET>
order deny,allow
deny from all
allow from .ncsa.uiuc.edu
</Limit>
```

*Protection by network domain -- exclusion.*

> This document[19] is accessible to clients running on machines anywhere *but* inside domain `ncsa.uiuc.edu`.
>
> *Note for NCSA readers:* The `.htaccess` file used in this case is as follows:

```
AuthUserFile /dev/null
AuthGroupFile /dev/null
AuthName ExampleDenyFromNCSA
AuthType Basic

<Limit GET>
order allow,deny
allow from all
deny from .ncsa.uiuc.edu
</Limit>
```

---

## For More Information

- NCSA HTTPd Authentication Directives AccessFileName[20] – Access Configuration[21]
- W3C on HTTP-based Authentication[22]
- RFC 1321 on MD5 Digest Algorithm[23]
- Draft IETF specification for MD5 Digest Authentication[24]
- Spyglass' Efforts on WWW Security[25]

---

Return to the Tutorial Overview[26]

[19] /examples/auth/by-domain-not-ncsa/access-me.html
[20] ../setup/srm/AccessFileName.html
[21] ../setup/access/Overview.html
[22] http://www.w3.org/hypertext/WWW/AccessAuthorization/Overview.html
[23] http://ds.internic.net/rfc/rfc1321.txt
[24] http://ds.internic.net/internet-drafts/draft-ietf-http-digest-aa-0 1.txt
[25] http://www.spyglass.com/six/developers_tech_doc2.html
[26] index.html

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[27] / 9−27−95*

[27] ../FeedBack.html

# WAIS and HTTP Integration

## Introduction

This document overviews existing methods for using WAIS as a back–end search engine for HTTP servers.

Information herein is currently experimental and may or may not work for you.

---

## WAIS and Plexus

Plexus[1] is a powerful Perl[2]–based HTTP server written and maintained by Tony Sanders[3] at BSDI[4].

---

## WAIS and WN

WN[5] is a multi–protocol server written and maintained by John Franks at NWU. It is shipped with support for WAIS as a back–end search engine[6].

---

## WAIS and NCSA HTTPd 1.0

Rob McCool has written a CGI script which allows NCSA HTTPd 1.0 as well as other CGI compliant servers to access a WAIS database in the same way that is mentioned in this document. The script is in the CGI archive[7]. It contains instructions for setting it up under HTTPd 1.0.

---

## freeWAIS 0.202's URL Type

freeWAIS 0.202 is shipped with support for type "URL". Use of this type is a little tricky.

First, **Mosaic 2.0 doesn't know how to deal with this type directly**, but Mosaic 2.1 (when it is released) will.

Second, use of this type apparently implies overloading the "headline" of a WAIS hit with the URL. This is fine, except then the description that the user sees of a given document *is* the URL, and URLs are, as usual, pretty cryptic things to just throw in front of average users.

But anyway, here's how it works:

```
waisindex ... -t URL what-to-trim what-to-add ...
```

1 http://www.bsdi.com/server/doc/plexus.html
2 http://www.cis.ufl.edu/perl/
3 http://www.bsdi.com/hyplan/sanders.html
4 http://www.bsdi.com/
5 http://hopf.math.nwu.edu/
6 http://hopf.math.nwu.edu:70/0h/docs/waisgn.guide
7 ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/cgi/wais.tar.Z





So what does that mean?

Well, first, `-t URL` tells `waisindex` to use type `URL` (note use of *lowercase* `-t` in this instance).

Second, `what-to-trim` and `what-to-add` are parameters that tell the indexer how to put together the URL that's returned as the result of a query.

Suppose your documents are normally stored in `/X11/mosaic/public`. Suppose also that these documents are normally served via a URL that begins with `http://hoohoo.ncsa.uiuc.edu:8080`.

> This means that a file stored as `/X11/mosaic/public/foo.html`, for example, is normally served as `http://hoohoo.ncsa.uiuc.edu:8080/foo.html`.

The `waisindex` command you'd use in this case would be something like the following:

```
waisindex -d ~/localwais/sources/www -export
  -t URL /X11/mosaic/public http://hoohoo.ncsa.uiuc.edu:8080
  /X11/mosaic/public/*.html
```

... where `~/localwais/sources/www` is the name of the WAIS index file and `/X11/mosaic/public/*.html` are the files you are indexing.

When queries are made on this database, the string `/X11/mosaic/public` is removed from the beginning of the filename of a matching file and the string `http://hoohoo.ncsa.uiuc.edu:8080` is put in its place.

> As per our previous example: `/X11/mosaic/public/foo.html` turns into `http://hoohoo.ncsa.uiuc.edu:8080/foo.html` as the result of a WAIS hit.

As you can see, this is perfect –– the WAIS server passes back the exact same URL that would normally be used to access this file via HTTP. So, everything from relative hyperlinks to relative inlined image references in the file will work correctly when the file is retrieved.

---

Return to the tutorial index[8]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[9] / 9–11–95*

8 index.html
9 ../FeedBack.html

# The New DirectoryIndex directive

## Purpose

When a client requests a directory, HTTPd can return a pre−written index, or generate one from the filesystem. The DirectoryIndex directive sets the file HTTPd should look for as a prewritten index to a given directory.

## Syntax

`DirectoryIndex file1 file2 file3`

Where `file1,file2,file3` are file names.

Only one DirectoryIndex directive is allowed in the server configuration file, but multiple files are allowed on the line. NCSA HTTPd will search for the listed files in order.

## File

srm.conf

## Default

If you do not specify a DirectoryIndex, HTTPd assumes:

`DirectoryIndex index.html index.shtml index.cgi`

## Example

`DirectoryIndex .index.html home.html .index.shtml`

A request for `/dir/` would cause the server to look for the file `DocumentRoot`[1] `/dir/.index.html`. If found, the server would send it back to the client. Otherwise, it would look for `DocumentRoot`[2]`/dir/home.html`, and then `DocumentRoot`[3]`/dir/.index.shtml` which could be a server side include. If none of these exist, then HTTPd would create and return an index from the filesystem.

NCSA HTTPd Development Team
*httpd@ncsa.uiuc.edu*

[1] DocumentRoot.html
[2] DocumentRoot.html
[3] DocumentRoot.html

# HTTP 1.0 KeepAlive

## Purpose

One of the features of HTTP is its statelessness. Practice has shown, however, that a separate connection for every request invokes a great deal of overhead, as well as abnormally high server loads. This is particularly true for slow connections or connections that must go through many routers.

In response to this, NCSA HTTPd now allows for multiple requests to be handled over a single connection with browsers which support this feature. Using one such browser, NCSA Mosaic for Windows, we found a speed increase of up to 30% using this feature. Of course, the actual performance improvement in practice will depend on the particular connection.

---

## Syntax

Three new directives have been introduced to configure the keepalive feature.

```
KeepAlive on | off
```

Where the syntax "`on | off`" means that either "on" or "off" may be present. This directive allows the administrator to enable or disable keepalive.

```
KeepAliveTimeout N
```

Where `N` is the number of seconds to maintain the connection between requests. If N seconds passes since the last client request on the connection, the server will close the connection.

```
MaxKeepAliveRequests N
```

Where `N` is an integer indicating the maximum number of requests that the server will accept during a keepalive session. After the client has issued this many requests, the server closes the connection.

---

## File

httpd.conf

---

## Defaults

If you do not include the KeepAlive directive in httpd.conf, keepalive will not be enabled.

The default for KeepAliveTimeout is 10 seconds. However, experience and experimentation will probably be required to determine a "good" value.

The default for MaxKeepAliveRequests is 5. However, if you do not want a limit imposed, you can use a maximum value of 0, which will then allow any number of





requests during a keepalive session. For example:

```
MaxKeepAliveRequests 0
```

---

## Points to Consider

Since HTTPd is a pre−forking server, most requests under normal load are handled by persistent processes. If these processes are busy handling keepalive sessions, then additional processes may be initiated to deal with requests. If the number of processes reaches Maxservers[1], then future requests will be handled by forked, non−persistent processes. Thus, there is a chance that many long keepalive sessions will lead to a large number of processes. We are still studying the effect of keepalive on the overall behavior of HTTPd.

Shortly after implementing the keepalive feature in HTTPd, a similar mechanism known as session extension[2], was proposed by the IETF for HTTP1.1. The mechanics are similar to our current implementation, but there are differences. We will most likely be modifying our keepalive implementation to be compatible with the session extension proposal during this beta. This should, for the most part, be transparent to the administration of HTTPd.

For Developers:

- The IETF Draft HTTP 1.0 Protocal[3]
- The IETF Draft on HTTP Session Extension[4]

[1] /docs−1.5B/setup/httpd/MaxServer.html
[2] http://www.ics.uci.edu/pub/ietf/http/draft−ietf−http−ses−ext−00.txt
[3] http://www.w3.org/hypertext/WWW/Protocols/HTTP1.0/draft−ietf−http−spec.html
[4] http://www.ics.uci.edu/pub/ietf/http/draft−ietf−http−ses−ext−00.txt

# The New Redirect Directives

## Purpose

The Redirect directives create virtual documents on your server, and any accesses to it will be redirected to a new URL. They also allow people to follow documents that have moved.

Now Available:
**Redirect**
**RedirectTemp**
**RedirectPermanent**

## Syntax

```
Redirect virtual URL
RedirectTemp virtual URL
RedirectPermanent virtual URL
```

`virtual` is the translated location which should trigger a redirect..

`URL` is the URL[1] of the new document.

Several Redirect directives may appear in the configuration file.
The Redirect directives are allowed in all Resource Configuration files (srm.conf) and Per Directory Configuration files (.htaccess).

## Example

```
Redirect /dir1 http://newserver.widget.com/dir1
```

This would cause requests for /dir1 to be redirected to the new location, `http://newserver.widget.com/dir1`.

## Specifics

In the Resource Configuration file (srm.conf), Redirects do a substring redirection, so that if you use the example above, any request for `/dir1/wow` would be redirected to `http://newserver.widget.com/dir1/wow`.

In the By Directory files (.htaccess), Redirects do a full string comparison, such that in the example above, only requests to `/dir1` **exactly** will be redirected to `http://newserver.widget.com/dir1`.

In Multihome support, Redirects are supported in the default srm.conf file, as well as other configuration files. If you don't give any Alias/ScriptAlias/Redirect directives in the specific resource file, the ones from the default will be used. If you give any of

[1] http://www.ncsa.uiuc.edu/demoweb/url−primer.html





the above in the specific resource file, then only those will be available for that hostname.

Redirect and RedirectTemp return 302 Temporary Redirects (as Redirect has allows done). RedirectPermanent returns 301 Permanent Redirect.

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−6−95*

# Enhanced Access Control

## Purpose

HTTPd has had the ability for some time to limit access to documents by either requiring users to be from certain domains, or to know a password. However, if you combined these features, the user would have to meet all the criteria. If you wanted people in your domain to have access, but limit outside access to users who have a password, people in your domain would be inconvenienced by having to authorize themselves with a password also.

In HTTPd 1.5, access to documents can be controlled in a more flexible manner. You can require that users be from certain domains and know a password, as in the past. In addition, you can now specify that users be from a certain domain, or know a password. This way local users are not inconvenienced by having to authorize themselves. Furthermore, you can use both methods in a particular document tree, using the AND behavior in some directories, and the OR behavior in others.

---

## Syntax

To enable the more flexible access control, a new directive has been added to the family of order[1], allow[2], and require[3] directives. This new directive is dubbed satisfy, and is used inside the Limit sectioning directive. The syntax is:

`Satisfy all | any`

Where you specify `all` if you want users to meet the restrictions of both allow and require directives, and you specify `any` if you only require that users meet one of the specified allow or require directives.

---

## File

- access.conf
- .htaccess files

---

## Default

If you do not use the Satisfy directive, the default is the AND behavior, i.e., users must satisfy an allow directive AND a require directive if both are present.

---

## Examples

< Limit GET>

order deny,allow

[1] /docs−1.5B/setup/access/order.html
[2] /docs−1.5B/setup/access/allow.html
[3] /docs−1.5B/setup/access/require.html





deny from all

allow from .ncsa.uiuc.edu

require user john stan lupe

satisfy any

< /Limit>

In the above example, anyone from domain ncsa.uiuc.edu can gain access to documents in the particular directory. In addition, users john, stan, and lupe can gain access, if they know their pre−arranged password.

Refer to the documentation on the Limit[4] directive for further information on the preceding example.

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−20−95*

[4] /docs−1.5B/setup/access/Limit.html

# DBM File Support

## Purpose

NCSA HTTPd 1.4 and before handled groups and passwords in a flat file database reminiscent of the /etc/passwd and /etc/group files found on most versions of un*x (especially older ones). This meant that lookups had to search through the file in order, comparing each entry until the match occured. While this was good enough for fewer then a hundred entries, by the time you hit 10000 entries, access times would be through the roof.

In order to alleviate this bottleneck, NCSA HTTPd 1.5 supports DBM files. DBM is a fairly straight forward database library that is supported on most un*x platforms. And even if your platform doesn't have it, chances are the GNU gdbm will compile on your platform, and you can use that. With DBM support, lookup times should be consistent, and consistently faster then under a comparable flat file.

With the DBM format, however, you lose some of the ease of use that was associated with the hand editable flat file. We have provided support programs to convert from one format to the other, as well as add/delete/modify DBM files without conversion.

---

## Syntax

There are three directives pertinent to DBM support: AuthUserFile, AuthGroupFile, and AuthDigestFile. These directives take one argument which specifies the full pathname of the particular file. For DBM support, these directives have been augmented to accept an optional second argument. If the filenames are followed by the word "dbm", HTTPd will assume that the files are in DBM format. If the filenames are followed by the word "standard", or no third argument is present, the standard flat file format will be assumed. Here are some examples of the syntax:

```
AuthUserFile /usr5/guillory/public_html/.htpasswd dbm
```

```
AuthGroupFile /usr5/guillory/public_html/.htgroup
standard
```

Note that you can different formats for your group and password files. In the above example, the word standard was unnecessary, as the standard format is the default.

---

## Support

There are several utilities in the support directory to manipulate DBM authentication files. The pair of utilities std2dbm.c and dbm2std.c are provided to convert back and forth between DBM and standard flat file formats. The std2dbm.c program can convert group files, password files, and digest files to DBM format. However, there is a current limitation on line length in group files. Lines in the .htgroup files must be no longer than 1023 characters. We will remove this limitation during this beta.

There are three utilities to manipulate DBM files directly: dbmpasswd.c, dbmgroup.c, and dbmdigest.c. These programs allow you to add entries, modify entries, and delete entries.





---

The Library:

- GNU's Free DBM: gdbm−1.7.3.tar.gz[1]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−06−95*

[1] ftp://prep.ai.mit.edu/pub/gnu/gdbm−1.7.3.tar.gz

# Changes Made During the 1.5 Beta Test

## Fixes for 1.5 Beta 6

- Added SERVER_ADMIN CGI Variable per requests
- Added LOCAL hack to allow matching in domain restriction of all local hostnames (Maurizio Codogno (mau@beatles.cselt.stet.it))
- Added support for HTTP/1.1 Host: header for virtual host support
- Various includes file and portability fixes
- Fixed weird special case of domain restriction failure from access.conf in conjunction with having .htaccess files
- Fixed new VirtualHost ErrorDocument support so as not to cause SIGSEGV
- Moved RefererIgnore matching to log_transaction() to allow HTTP/1.1 host support to work
- Changed debugging dump (SIGUSR1)
- Typo in ErrorDocument settings
- Added Optional and Required parameters to VirtualHost configuration tag to allow for a single host configuration failure not to bring down all of them
- Added patch by Elf Sternberg (elf@aaden.spry.com) to allow configuring of process titles in SETPROCTITLE
- Change LogType to LogOptions and added a new option (ServerName) which will log the servername of the request to the access_log
- Added CoreDirectory directive to tell HTTPd where to dump core

## Fixes for 1.5 Beta 5

- Added support for Combined LogType (CLF with referer and useragent)
- Added check at launch to attempt to keep people from leaving a security hole open by leaving the log file directory writeable by the wrong people
- Added support for internal imagemap support
- Added support for VirtualHost level ErrorDocuments
- Added date stamp to UserAgent and RefererLogs to allow for easier correlation
- Added file descriptor wrappers for directories and DBM files
- Fix SSI bug where index filename not added to DOCUMENT_URI
- Don't pass PATH_INFO information to index files unless specifically refered to (ie, do in case of /index.cgi/wow, but not in /wow)
- Add "fix" by Sun for better performance under Solaris
- Change children proctitle to "idle"
- Removed getdomainname patch, its not portable and not the right thing to do
- changed atoi() in uname2id() and gname2id() to scan_long. If the atoi failed, it would return 0, causing the server to run as root

## Fixes for 1.5 Beta 4

- Fixed typo in FD_LINUX support
- Fixed support for NeXT platform
- order of operations wrong on restart
- Now pass environment to CGI script from SSI
- Fix Alias/ScriptAlias/Redirects for VirtualHost
- Fix if all of POST data read on first pass, don't try to read more as seen through CERN proxy
- Work around for systems which don't support numeric IP addresses passed to





gethostbyname()

- Added log_reason for non−exec CGI
- Pass reqInfo to init_header_vars() in local redirects from cgi scripts
- use unbuffered i/o for file writes to fix fwrite bug on AIX 3.2.5 for PowerPC
- fix write loop for send_script() for CGI scripts
- use stat inplace of lstat to fix IfSymLinkOwner bug if link to link to file
- should now recognize and use numeric IP on systems with older versions of bind which don't support a numeric IP to gethostbyname()
- Fix SSI, flastmod wasn't working
- freeing ENV array at the wrong time, esp. in SSI
- Fix refererignore again
- Ultrix not brain dead since 4.4, so put #define in old hack for determining which version of ultrix you are compiling on
- possible help for get_local_host() on some systems which didn't work before
- HEAD_GETPASS for ultrix in support directory

## Fixes for 1.5 Beta 3

- Fixes to various scripts for cleaner compiles
- Added KeepAlive directives to example httpd.conf
- Fixed Makefile support for SVR4, added ConvexOS, QNX, ATTSVR3 from 1.4.2
- Cleaned up header include directives for portability
- Added file descriptor wrapper to prevent leaks
- Cleaned up exec code, so forked child always dies even if exec() fails in a funny way (should make sure no children of children get left around)
- file descriptor leak if CGI script returns Location header closed
- attempt to keep environment and authorization between multiple requests like for internal redirects and on ErrorDocument
- Added NIS_SUPPORT patch by Kevin Ruddy
- Fixed Multiple referer ignore
- fixed an off by one error in environment handling (env.c). Probable cause of coredumps on Solaris machines, possibly others.
- Now allow DocumentRoot in httpd.conf file (for VirtualHosts)
- Added support for Linux file descriptor passing /proc hack
- Cleaned up File Not Found error message (per HyperNews suggestion)
- Allow restart when relative path used with −f on command line Vince Tkac (tkac@oclc.org)
- Cleaned up Restart code ( kill −HUP ). Should now work on all systems regardless of optimization
- Cleaned up sockaddr usage
- Moved socket initialiazation to its own function
- Now log start command to error_log
- Improved usage function (for −v command line)
- Made sigjmp_buf default define for JMP_BUF (missed from 1.4.2)
- Fixed getline()

## Changes between 1.5 Beta 1 and 1.5 Beta 2

- To fix AddType .cgi completely, now has a single virtual path for any method to a "standard document" and checks just before sending if the method is valid for the document type.
- General code reorganization, including renaming of scripts to cgi and removing all method specific files, and creating an http_send for the new general case

- Moved all non−security compile time options to config.h
- Support for profiling the code added
- A/UX, SVR4, SCO3 support updated
- New type of process title changing subroutine added, and made a compile time option
- Collapsed number of malloc()'s for performance
- DBM support now properly #define'd
- Duh, always initialize url/args/filename
- fixed multihome support to allow more than 2 entries
- fixed access control
- log missing user attempt − Gioacchino La Vecchia (gio@di.unipi.it)
- explicitly allow or disallow certain server configuration directives in virtual hosts
- strip ending / on document root at configuration time
- stat filesystem for .htaccess files instead of checking failure of fopen() for speed improvement
- change order of mime initialization to get actual location of file first
- missing LF in redirect message from die()
- actually log the error message if using an error document for 500 errors
- close more file descriptors on errors
- make separate content_type variables for type as sent and type of reply
- allow Connection header to be made into a CGI variable
- fix signal handling during CGI
- Only shutdown(sd,0) to disallow receives not sends and receives This fix was for OSF/1 2.0, but may fix AIX problems of garbling ends of images
- attempt to lessen the number of fflush()'s called during a transaction
- move local variables that might be clobbered by setjmp/longjmp
- call tzset on all systems
- log errno on some errors
- log which URL is being aborted or timing out
- Keberos fixes
- Multiple referer ignore added
- Probably some we've missed

---

Return to 1.5 Beta Test Home Page[1]

Last Modified 10−06−95

[1] /beta−1.5/index.html

# Changes Made During the 1.5 Beta Test

## Changes between 1.5 Beta 2 and 1.5 Beta 3

- Fixed a file descriptor leak associated with location headers sent from scripts.
- Fixed an array overflow problem with the environment variables that caused seg faults primarily on solaris systems.
- Beta 2 broke referrer ignore trying to implement multiple referer ignore. This was fixed.
- Patch for file descriptor passing in linux added.
- Put a system in place to track file descriptor usage. This should allows us to better clean up after exception conditions such as user aborts.
- Support for NIS authorization.
- Fixed a bug that cropped up when multiple reads were required on a socket/pipe to receive all the data.
- General code reorganization, including renaming of scripts to cgi and removing all method specific files, and creating an http_send for the new general case
- Probably some we've missed

---

Return to 1.5 Beta Test Home Page[1]

Last Modified 8–09–95

[1] /beta–1.5/index.html

# Kerberos Authentication

## Purpose

Kerberos is a "trusted third party" authentication system developed at MIT which allows clients and servers to authenticate and communicate securely over an untrusted network. It uses symmetric−key encryption technology (DES) to avoid passwords being sent over the network in the clear. For an introduction to Kerberos, please see:
Kerberos: An Authentication Service for Open Network Systems [1]
or
The Kerberos FAQ [2]

Using kerberized web browsers and servers have the following advantages:

- passing kerberos tickets allows for authentication without passwords
- after a client user logins to the system (via "kinit"), all subsequent kerberos ticket operations happen automatically (transparent to the user).
- in addition to specifying precisely which users may view certain web documents, the server admin may also allow access to any kerberos−authenticated user. This can be particularly useful for admins who wish to make use of an institution's main kerberos database and not replicate large .htpasswd files.

Note: Kerberos V4 and V5 may be used simultaneously, in case you are in two realms (perhaps as a part of V4−>V5 migration).

## Compiling httpd with Kerberos support

You will need to have the relevant Kerberos libraries and include files installed on your machine. So set the "KRB4DIR" and/or "KRB5DIR" defines in the httpd's Makefile (in the "src" directory) appropriately. Then be sure to un−comment the "KRBFLAGS" and "KRBLIBS" lines in this Makefile, since the default settings do not include Kerberos support.

You can compile httpd with support for Kerberos V4 *and/or* Kerberos V5 (yes they will coexist peacefully, if need be). If you only want one flavor, simply comment out the lines related to the other flavor. For example, if you want V5 support but not V4, comment out the Makefile lines for "KRB4DIR", "KRB4FLAGS", and "KRB4LIBS".

## Syntax of .htaccess file

To require Kerberos authentication for viewers of a particular document directory, you must assign `AuthType` to either `KerberosV4` or `KerberosV5`. Then you can use the require lines to further specify allowed users.

[1] ftp://athena−dist.mit.edu/kerberos/doc/usenix.PS
[2] http://snapple.ncsa.uiuc.edu/adam/khttp/KERBEROS.FAQ





- **require user**

  You may grant access to particular users by specifying either their full kerberos principal name or just their username (the first field of their principal name). In the latter case, the user is assumed to in the same realm as the server.

  For instance, assume your server is in the realm `FOO.BAR.EDU` . The line:

  ```
  require user ted alice fred@SOME.OTHER.REALM
  ```

  will grant access to principals `ted@FOO.BAR.EDU`, `alice@FOO.BAR.EDU`, and `fred@SOME.OTHER.REALM` . The latter case assumes that you have a cross−realm mechanism established for `SOME.OTHER.REALM`.

- **require valid−user**

  This line will allow access to any authenticated user in the server's realm.

- **require realm**

  For more sophisticated cross−realm applications, you may grant access to any authenticated user in particular realms using `require realm` lines. For instance, the line:

  ```
  require realm LANL.GOV ANL.GOV
  ```

  will allow access to any authenticated user in the `LANL.GOV` or `ANL.GOV` realm.

## Additional Configuration Steps

**Make sure Kerberos is properly configured on the server machine**

This means that your Kerberos configuration file(s) are installed and contain the appropriate realm name, host−>realm mapping and such. For Kerberos V4, this involves the "krb.conf" and "krb.realms" files (usually in the "/etc/athena" directory); for Kerberos V5, this only involves the "krb5.conf" file (usually in "/etc").

**Defining the "khttp" principal:**

To use the kerberos features in httpd, your realm must have a principal defined for the "khttp" service on the server's machine. So if your realm is `FOO.BAR.EDU` and you want to run the kerberized httpd on a machine named `howard.foo.bar.edu` , there will need to be a kerberos principal named:

- `khttp.howard@FOO.BAR.EDU` (for Kerberos V4)
  or
- `khttp/howard.foo.bar.edu@FOO.BAR.EDU` (for Kerberos V5)

Note: if you have a DNS alias mapping the name "www.foo.bar.edu" to "howard.foo.bar.edu", you would want to use the "www" in your principal name. Also note: there may be other ways of handling the instance field (the one with the server's host name) which work. The method explained above is the one that has been working well in our tests so far.

For details on how to add a new principal to your Kerberos database, please see the documentation that comes with your Kerberos package.

**Setting up the server's keyfile**

Once you have the "khttp" principal defined, you must make that principal's secret key available in a keyfile on the server's machine. To do this, a "srvtab" must be created using the Kerberos admin utilities ("kdb5_edit", for V5; "make_srvtab" or "ksrvutil" for V4). This srvtab file must be saved in a sensible location on the machine running httpd (see the "Defaults" section below). When you run httpd, you can (currently, you must) give it the srvtab filename(s) on the command line using the "−k" and/or "−K" options.

`httpd -k /etc/athena/srvtab` sets the Kerberos V4 keyfile to "etc/athena/srvtab"

`httpd -k /etc/v5srvtab` sets the Kerberos V5 keyfile to "etc/v5srvtab"

*Important:* **Set the file permissions carefully**

The security of your kerberized server depends on the handling of the keyfile. This file should only be readable by the web daemon (as well as "root"). Installing the keyfiles in the standard places (/etc) will probably require being root. However, it is unwise to have httpd run as root (see the options for "httpd.conf"). Here is one way to do things:

- Run httpd as a special user, perhaps called "web"
- Create a group, perhaps called "kservers", containing members "root" and "web".
- Install the keyfile, making it group−readable and chgrp it to group "kservers".
- Make sure the keyfile is not world−readable.
- So now, the keyfile is owned by root and readable by members of group "web", which includes the httpd username.

Alternatively, you could (as root) simply change the ownership of the keyfile to "web" and make it only readable by owner.

---

## Files

- /etc/athena/srvtab (V4 default keyfile)
- /etc/v5srvtab (V5 default keyfile)

- /etc/athena/krb.conf (V4 default configuration file)
- /etc/athena/krb.realms (V4 default host−>realm mapping file)
- /etc/athena/krb5.conf (V5 default configuration file)

---

## Defaults

The httpd binaries are compiled without Kerberos features. The Makefile that comes with the source distribution also has Kerberos disabled, as a default.





Note that currently (beta 1), you must give the filename of the server's key as a command line argument to httpd to make the kerberos features work. The default values for these filenames may not work (yes, a bug). In the future, this may become a configuration option. Please tell us how you would like this file specified.

---

## Examples

Here is a sample .htaccess file in some document directory:

```
AuthType KerberosV4
< Limit GET>

order deny,allow
deny from all
allow from .ncsa.uiuc.edu
require valid-user
< /Limit>
```

This grants access to any authenticated client in the server's Kerberos realm who are connecting from a machine in the ".ncsa.uiuc.edu" domain (double protection, if you will).

---

## For More Information

Web Kerberization Intro [3]

If you have problems or questions, send email to the NCSA httpd group's Kerberos dude,
*acain@ncsa.uiuc.edu*

comming soon: documentation on using Kerberized Mosaic
in brief:

- Use a Kerberized browser (e.g., Xmosaic 2.7b1)
- Browsers supporting Columbia−style Kerberos V4 support (patcheds by Ben Fried) should also work
- Make sure the client machine's Kerberos configuration files are set appropriately for client operations. For V4, check the "krb.conf" and "krb.realms" files; for V5, check the "krb5.conf" file.
- Make sure your /etc/services file has the ports defined for Kerberos (see the Kerberos installation guide for details).

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−21− 95*

[3] http://snapple.ncsa.uiuc.edu/adam/khttp/intro.html

# MD5 Authentication Support

## Purpose

The main problem with Basic ByPassword authentication is that on every transaction, your password and login are sent over the net uuencoded. UUencoding is not much of a deterrant to those wily crackers, so the folks at Spyglass[1] came up with MD5 authentication.

MD5[2] is a mathematical formula that is supposed to have a very small probability of giving duplicate strings for different inputs. Using the password the user types in, the browser would create a string using the password and other information including the AuthRealm, and pass it through MD5. This string is then passed over the net to the server, which takes the information stored in the .htpasswd file to create the same string and pass it through MD5, and then compare the results. This way, it is nearly impossible to get the password out of the information sent across because the password isn't sent across.

Sounds to good to be true? Well, there is a hitch. While its safer to use across the network, it is more difficult to make safer on the local machine. Under Basic auth, the passwords are passed through crypt() and stored. crypt() is a one way function, and the passwords can be viewed with little chance of being compromised (assuming the passwords are fairly unlikely to be guessed, since the only way to compromise them is the try crypt()'ing strings and comparing to the result). Under MD5, we need the information stored in the .htpasswd file, so we can't use a one way crypt() on it. Although we use MD5 on the password before storing it, if you have the stored code, and a little knowledge, you can create a browser to use it instead of the password, and it would work.

The solution is to enforce the unix file permissions. Most servers these days run under an unprivileged user and group, and in order for anything to be viewable to the server, they have to be world readable. The best fix, on platforms which support it, is to set the group on all of the directories (and files in them) to a group which the server is in, and the users are not. Then, set the group id bit on the directories (chmod g+s). This should make it so that every file created in the directory is grouped in that group. Make everything rw for user and r for group, and you should be on your way. We're hoping that this beta testing will help us write a more informative guide to doing this.

---

## Syntax

```
AuthDigestFile /u/Web/.htdigest
```

```
AuthType Digest
```

The AuthDigestFile is a new directive that takes one argument, the full pathname of the file in which the digest authentication data is stored in.

To indicate that a directory is protected with MD5, you specify an AuthType of Digest, as shown above.

1 http://www.spyglass.com
2 http://ds.internic.net/rfc/rfc1321.txt





## File

access.conf
.htaccess

---

## Support Tools

There is a utility in the support directory called htdigest. This tool is helpful in creating the .htdigest files that MD5 needs. A more convenient version of this tool will be made available during this beta. It will have an interface similar to the htpasswd utility.

---

For developers:

- RFC 1321 on MD5 Digest Algorithm[3]
- Draft IETF specification for MD5 Digest Authentication[4]
- Spyglass' Efforts on WWW Security[5]

---

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu / Last Modified 7−06−−95*

3 http://ds.internic.net/rfc/rfc1321.txt
4 http://ds.internic.net/internet−drafts/draft−ietf−http−digest−aa−01.txt
5 http://www.spyglass.com/six/developers_tech_doc2.html

# NCSA HTTPd Postscript Documentation

The documents online have the advantage of being able to run demos or following plenty of inter−connecting links,which is not supported by these postscript versions. To see a demo, or if something is not made quite clear through the postscript docs, the user should consult the online docs for a possible clarification.

- **Introduction To Server**[1]
- **Installation Instructions**[2]
- **Using the Server**[3]
- **Tutorials**[4]

---

These are some extra documents that might be of interest.

- **Upgrade Notes**[5]
- **Example Access Configuration File**[6]

---

Return to the Overview[7]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[8] / 09−11−95*

[1] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/documents/IntroToServer.ps.Z
[2] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/documents/Install.ps.Z
[3] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/documents/usage.ps.Z
[4] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/documents/tutorial.ps.Z
[5] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/documents/upgrades.ps.Z
[6] ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/documents/accessconf.ps.Z
[7] Overview.html
[8] FeedBack.html

# NCSA HTTPd Acknowledgements

Thanks to:

Robert McCool[1]
For developing NCSA HTTPd till version 1.3[2] and this documentation.
Carlos Varela[3]
For HTTPd and CGI support/development up to and including NCSA HTTPd 1.4
Brandon Long[4]
For development of NCSA HTTPd 1.4 (and beyond)
Yuxin Zhou[5]
For HTTPd support for NCSA HTTPd 1.4
Stanford Guillory
For development of NCSA HTTPd 1.4 (and beyond)
Carlos Pero[6]
For CGI support
Adam Cain[7]
Security support/development
Beth Frank[8]
NCSA HTTPd Development Team Manager

## A Note to Users

- Any questions or bug reports regarding NCSA HTTPd should go to *httpd@ncsa.uiuc.edu*.[9]
- Any questions or bug reports regarding CGI should go to *cgi@ncsa.uiuc.edu*.[10]

*NCSA HTTPd Development Team / httpd@ncsa.uiuc.edu[11] / Last Modified 7–14–95*

[1] http://home.netscape.com/people/robm/
[2] http://home.netscape.com/people/robm/why–httpd–sucks.html
[3] http://www.uiuc.edu/ph/www/cvarela
[4] http://www.uiuc.edu/ph/www/blong
[5] http://www.uiuc.edu/ph/www/z–yuxin
[6] http://www.uiuc.edu/ph/www/c–pero
[7] http://snapple.ncsa.uiuc.edu/adam.html
[8] http://sdg.ncsa.uiuc.edu/~efrank/
[9] FeedBack.html
[10] /cgi–bin/mail?cgi
[11] FeedBack.html