

[FAQ](#)[SiteMap](#)[Directives](#)[Modules](#)[Search](#)

Apache HTTP Server Version 1.3

Release Notes

[New Features in Version 1.3](#)[Upgrading to Version 1.3](#)[Apache License](#)

Reference Manual

[Compiling and Installing](#)[Starting](#)[Stopping or Restarting](#)[Run-time Configuration Directives](#)[Modules: \[By Type\]\(#\) or \[Alphabetical\]\(#\)](#)[Server and Supporting Programs](#)[Dynamic Shared Object \(DSO\) Support](#)[The Apache API](#)

Platform Specific Notes

[Microsoft Windows](#) | [Cygwin](#)[Novell Netware 5](#)[HP MPE/iX](#)[UnixWare](#)[TPF](#)

Using the Apache HTTP Server

[Authentication, Authorization, and Access Control](#)[CGI: Dynamic Content with CGI](#)[Configuration Files](#)[Content negotiation](#)[Environment Variables](#)[General Performance hints](#)[Handlers](#)[Log Files](#)[Security tips](#)[Server Side Includes](#)[Server-Wide Configuration](#)[suexec: Using SetUserID Execution for CGI](#)[URL Mapping: Mapping URLs to the Filesystem](#)[URL Rewriting Guide](#)[Virtual Hosts](#)

Other Topics

[Frequently Asked Questions](#)[SiteMap](#)[Tutorials](#)[Other Notes](#)

Maintained by the [Apache HTTP Server Documentation Project](#).

Apache HTTP Server





Apache HTTP Server

Site Map

- [Apache HTTP Server Version 1.3 Documentation](#)
 - Release Notes
 - [Upgrading to 1.3 from 1.2](#)
 - [New features with Apache 1.3](#)
 - Using the Apache HTTP Server
 - [Compiling and Installing Apache](#)
 - [Starting Apache](#)
 - [Stopping and Restarting Apache](#)
 - [Configuration Files](#)
 - [How Directory, Location and Files sections work](#)
 - [Server-Wide Configuration](#)
 - [Log Files](#)
 - [Mapping URLs to Filesystem Locations](#)
 - [Security Tips](#)
 - [Dynamic Shared Object \(DSO\) support](#)
 - [Apache Content Negotiation](#)
 - [Custom error responses](#)
 - [Setting which addresses and ports Apache uses](#)
 - [Environment Variables in Apache](#)
 - [Apache's Handler Use](#)
 - [suEXEC Support](#)
 - [Performance Hints](#)
 - [URL Rewriting Guide](#)
 - [Apache Virtual Host documentation](#)
 - [Name-based Virtual Hosts](#)
 - [IP-based Virtual Host Support](#)
 - [Dynamically configured mass virtual hosting](#)
 - [VirtualHost Examples](#)
 - [An In-Depth Discussion of Virtual Host Matching](#)
 - [File Descriptor Limits with Virtual Hosts](#)
 - [Issues Regarding DNS and Apache](#)
 - [Apache Server Frequently Asked Questions](#)

- Guides, Tutorials, and HowTos
 - [Authentication, Authorization, and Access Control](#)
 - [Apache Tutorial: Dynamic Content with CGI](#)
 - [Apache Tutorial: Introduction to Server Side Includes](#)
 - [HOWTO documentation](#)
 - [Apache Tutorials](#)
- Platform-specific Notes
 - [Using Apache with Microsoft Windows](#)
 - [Compiling Apache for Microsoft Windows](#)
 - [Running Apache for Windows as a Service](#)
 - [Using Apache with Cygwin](#)
 - [The Apache EBCDIC Port](#)
 - [The Apache TPF Port](#)
 - [Installing Apache on TPF](#)
 - [Using Apache with HP MPE/iX](#)
 - [Using Apache with Novell NetWare](#)
 - [Compiling Apache under UnixWare](#)
 - [Running a High-Performance Web Server for BSD](#)
 - [Performance Tuning Tips for Digital Unix](#)
 - [Running a High-Performance Web Server on HPUX](#)
 - [Hints on Running a High-Performance Web Server](#)
- [Apache HTTP Server and Supporting Programs](#)
 - [Manual Page: httpd - Apache HTTP Server](#)
 - [Manual Page: apxs - Apache HTTP Server](#)
 - [Manual Page: ab - Apache HTTP Server](#)
 - [Manual Page: apachectl - Apache HTTP Server](#)
 - [Manual Page: dbmmanage - Apache HTTP Server](#)
 - [Manual Page: htdigest - Apache HTTP Server](#)
 - [Manual Page: htpasswd - Apache HTTP Server](#)
 - [Manual Page: logresolve - Apache HTTP Server](#)
 - [Manual Page: rotatelog - Apache HTTP Server](#)
 - [Manual Page: suexec - Apache HTTP Server](#)
 - [Other Programs - Apache HTTP Server](#)
- [Apache modules](#)
 - [Apache modules - By Type](#)
 - [Apache directives](#)
 - [Definitions of terms used to describe Apache directives](#)
 - [Definitions of terms used to describe Apache modules](#)
 - [Apache Core Features](#)

- [Apache module mod_access](#)
- [Apache module mod_actions](#)
- [Apache module mod_alias](#)
- [Apache module mod_asis](#)
- [Apache module mod_auth](#)
- [Apache module mod_auth_anon.c](#)
- [Apache module mod_auth_db](#)
- [Apache module mod_auth_dbm](#)
- [Apache module mod_auth_digest](#)
- [Apache module mod_autoindex](#)
- [Apache module mod_cern_meta](#)
- [Apache module mod_cgi](#)
- [Apache module mod_digest](#)
- [Apache module mod_dir](#)
- [Apache module mod_env](#)
- [Apache module mod_example](#)
- [Apache module mod_expires](#)
- [Apache module mod_headers](#)
- [Apache module mod_imap](#)
- [Apache module mod_include](#)
- [Apache module mod_info](#)
- [Apache module mod_isapi](#)
- [Apache module mod_log_agent](#)
- [Apache module mod_log_config](#)
- [Apache module mod_log_referer](#)
- [Apache module mod_mime](#)
- [Apache module mod_mime_magic](#)
- [Apache module mod_mmap_static](#)
- [Apache module mod_negotiation](#)
- [Apache module mod_proxy](#)
- [Apache module mod_rewrite](#)
- [Apache module mod_setenvif](#)
- [Apache module mod_so](#)
- [Apache module mod_speling](#)
- [Apache module mod_status](#)
- [Apache module mod_unique_id](#)
- [Apache module mod_userdir](#)
- [Apache module mod_usertrack](#)
- [Apache module mod_vhost_alias](#)

- [Apache API notes](#)
- [Older Documentation](#)
 - [PATH_INFO Changes in the CGI Environment](#)
 - [Apache Keep-Alive Support](#)
 - [Apache Multiple Log Files](#)
 - [Server Pool Management](#)
 - [Source Re-organisation](#)
 - [Notes about Compatibility with NCSA's Server](#)
 - [International Customized Server Error Messages](#)
 - [Descriptors and Apache](#)
 - [Connections in FIN_WAIT_2 and Apache](#)
 - [Known Client Problems](#)
 - [Why We Took PEM Out of Apache](#)
 - [Configuring Multiple IP Addresses](#)
 - [MS Windows Netscape 3.0b4 KeepAlive problem solved](#)
 - [Reading Client Input in Apache 1.2](#)

Apache HTTP Server



```
/* =====
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000 The Apache Software Foundation. All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 * notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 * notice, this list of conditions and the following disclaimer in
 * the documentation and/or other materials provided with the
 * distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 * if any, must include the following acknowledgment:
 *     "This product includes software developed by the
 *     Apache Software Foundation (http://www.apache.org/)."
 * Alternately, this acknowledgment may appear in the software itself,
 * if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 * not be used to endorse or promote products derived from this
 * software without prior written permission. For written
 * permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 * nor may "Apache" appear in their name, without prior written
 * permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * =====
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation. For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing Applications,
 * University of Illinois, Urbana-Champaign.
 */
```



Apache HTTP Server

Upgrading to 1.3 from 1.2

In order to assist folks upgrading we are now going to maintain a document describing information critical to existing Apache users. Note that it only lists differences between recent major releases, so for example, folks using Apache 1.1 or earlier will have to figure out what changed up to Apache 1.2 before this document can be considered relevant. Old users could look at the `src/CHANGES` file which tracks code changes.

These are intended to be brief notes, and you should be able to find more information in either the [New Features](#) document, or in the `src/CHANGES` file.

Compile-Time Configuration Changes

- The source code has been [reorganized](#), which affects anyone with custom modules or modifications. But also, the `Module` directive has been changed to the `AddModule` directive.
- The Configuration variable `EXTRA_LFLAGS` has been renamed `EXTRA_LDFLAGS`.
- The `-DMAXIMUM_DNS` definition has been obsoleted by changes to `mod_access` enforcing double-reverse DNS lookups when necessary.
- The `-DSERVER_SUBVERSION=\"string\"` compile-time option has been replaced with the run-time API call `ap_add_version_component()`. Compile-time modification of the server identity by the configuration scripts is no longer supported.
- `mod_dir` has been split into two pieces [mod_autoindex](#), and [mod_dir](#).
- [mod_browser](#) has been replaced by [mod_setenvif](#).
- IRIX systems with untrusted users who can write CGIs which execute as the same uid as `httpd` should consider using `suexec`, or adding `-DUSE_FCNTL_SERIALIZED_ACCEPT` to `EXTRA_CFLAGS`. This is slower, more information is available on the [performance tuning page](#). There is a mild denial of service attack possible with the default config, but the default config is an order of magnitude faster.
- `mod_auth_mysql` has been removed from the distribution.
- The new Apache Autoconf-style Interface (APACI) was added to the top-level to provide a real out-of-the-box build and installation procedure for the complete Apache package.

Run-Time Configuration Changes

- There have been numerous changes to the default config files. Ensure that you compare your existing configuration files with the new ones to ensure there aren't any undesired differences. In particular:
 - As of Apache 1.3.0, the current config files apply different [Options](#) and [AllowOverride](#) settings to various directories than were used in 1.2.
 - As of the release following Apache 1.3.3, the three config file templates have been merged into `httpd.conf-dist` and the order of the directives changed.
- As of 1.3.2, [mod_expires](#) will add Expires headers to content that does not come from a file on disk, unless you are using a modification time based setting. Previously, it would never add an Expires header unless content came from a file on disk. This could result in Expires headers being added in places where they were not previously added.
- Standalone **FancyIndexing** directives are now combined with the settings of any `IndexOptions` directive already in effect, rather than replacing them.
- **AuthName strings will need to be quoted** in `.htaccess` or server configuration files if they contain blank characters (like

spaces). For example, if you use an `AuthName` directive like this:

```
AuthName This and That
```

you will need to change it to

```
AuthName "This and That"
```

This change was made for consistency in the config language.

- **As of Apache 1.3.1, methods listed in `<Limit>` directives must be uppercase.** Method names, such as GET, POST, and PUT are defined as being case-sensitive. That is, a GET request is different from a get request. Prior to Apache 1.3.1, the `<Limit>` directive parser incorrectly treated both of these as being the same. Apache's built-in method limit processing currently only understands uppercase method names, so if you've used clauses such as "`<Limit Get post>`" in your configuration files, you need to correct them to use uppercase names.
- Unrecognized method names in the server configuration files will result in the server logging an error message and failing to start. In `.htaccess` files, unknown methods will cause the server to log an error to its error log and return an 'Internal Server Error' page to the client.
- **The default Apache `ServerRoot` directory changed** from the NCSA-compatible `/usr/local/etc/httpd/` to `/usr/local/apache/`. This change covers only the default setting (and the documentation); it is of course possible to override it using the `-d ServerRoot` and `-f httpd.conf` switches when starting apache.
- Folks using HTTP/1.1-style virtual hosting will need to list the ip:port pairs that are supposed to have HTTP/1.1-style virtual hosting via the [NameVirtualHost](#) directive (one directive per pair). Previously this support was given implicitly on the "main server address". Now it has to be explicitly listed so as to avoid many problems that users had. Please see the [Apache Virtual Host documentation](#) for further details on configuration.
- The precedence of virtual hosts has been reversed (applies mainly to vhosts using HTTP/1.1 Host: headers, and the [ServerPath](#) directive). Now the earlier vhosts in the file have precedence over the later vhosts.
- `HostnameLookups` defaults to Off.
- **REMOTE_HOST CGI variable changed.** In Apache 1.2 and earlier, the `REMOTE_HOST` environment variable made available to CGI scripts was set to either the full DNS name of the client, or else to the client's IP address if the name was not known. This behavior differed from that specified by the CGI specification, which defines this variable as being NULL if the name isn't known. In Apache 1.3, we have made this correction. `REMOTE_ADDR` always contains the client's IP address, but `REMOTE_HOST` is only defined when the server has been able to determine the client's DNS name.
- The undocumented [mod_access](#) syntax "allow user-agents" was removed. The replacement is the more general "allow from env".
- When using wildcards in pathnames (such as * and ?) they no longer match / (slash). That is, they more closely behave how a UNIX shell behaves. This affects `<Directory>` directives, for example.
- If no `TransferLog` directive is given then nothing will be logged. (Previously it would default to `logs/access_log`.)
- Apache now has [configurable error logging levels](#), and the default eliminates some messages that earlier versions always generated.
- When booting, Apache will now detach itself from `stdin`, `stdout`, and `stderr`. `stderr` will not be detached until after the config files have been read so you will be able to see initial error messages. After that all errors are logged in the `error_log`. This makes it more convenient to start Apache via `rsh`, `ssh`, or `crontabs`.
- `<Files>` sections previously could take a full pathname, and were matched against the full pathnames. This had some inconsistencies, and was removed. To emulate this older behavior use a `<Files>` section nested inside a `<Directory>` section.
- `<Location>` matching behavior with respect to slashes has changed. See the [<Location> documentation](#) for more info.

Misc Changes

- `ServerType inetd` has been deprecated. It still exists, but bugs are unlikely to be fixed.
- `httpd_monitor` has been deprecated. The replacement is to use `mod_status` and make a request to a URL such as `http://myhost/server-status?refresh=10`.

- Apache now provides an effectively unbuffered connection for CGI scripts. This means that data will be sent to the client as soon as the CGI pauses or stops output; previously, Apache would buffer the output up to a fixed buffer size before sending, which could result in the user viewing an empty page until the CGI finished or output a complete buffer. It is no longer necessary to use an "nph-" CGI to get unbuffered output. Given that most CGIs are written in a language that by default does buffering (*e.g.*, perl) this shouldn't have a detrimental effect on performance.

"nph-" CGIs, which formerly provided a direct socket to the client without any server post-processing, were not fully compatible with HTTP/1.1 or SSL support. As such they would have had to implement the transport details, such as encryption or chunking, in order to work properly in certain situations. Now, the only difference between nph and non-nph scripts is "non-parsed headers".

- dbmmanage has been overhauled.

Third Party Modules

The following changes between the 1.2 and 1.3 API may require slight changes in third party modules not maintained by Apache.

- To avoid symbol clashes with third-party code compiled into the server, the general prefix ``ap_`` was globally applied to the following classes of symbols: Apache provided general functions (*e.g.*, `ap_cpystn`), public API functions (*e.g.*, `palloc`, `bgets`) and private functions which can't be made static (because of cross-object usage) but should be (*e.g.*, `new_connection`). For backward source compatibility with Apache 1.2 a new header file named `compat.h` was created which provides defines for the old symbol names. You'll either have to `#include compat.h` or update the API symbols you use.
- Be sure and examine the [source code reorganization page](#) to see whether any item there affects you.
- Use of `SERVER_VERSION` definition. If third-party modules reference the server version string using this symbol, they should be corrected to obtain it by calling the new API routine `const char *ap_get_server_version()`.
- `ap_construct_url` prototype change. The second parameter was previously a `server_rec`, it has been changed to a `request_rec`.
- The `table` datatype has been made an opaque type. Code which assumes a `table` is the same as an `array_header` will not compile. This is actually a change to enforce the API the way it was intended, all versions of Apache have had a `table_elts()` function which is intended for code which needs to access the elements of a table. The changes required for this are pretty easy, and work with all versions of Apache.

Suppose `t` is a table. Whenever code refers to `t->elts`, replace it with something like this:

```
array_header *arr = table_elts(t);
table_entry *elts = (table_entry *)arr->elts;
```

Whenever code refers to `t->nelts` use `arr->nelts`. Many examples can be found in the standard modules, search for `table_elts`.

Apache HTTP Server





Apache HTTP Server

Overview of New Features in Apache 1.3

New features with this release, as extensions of the Apache functionality. Because the core code has changed so significantly, there are certain liberties that earlier versions of Apache (and the NCSA daemon) took that recent Apache versions are pickier about - please check the [compatibility notes](#) if you have any problems.

If you're upgrading from Apache 1.2, you may wish to read the [upgrade notes](#).

Enhancements: [Core](#) | [Performance](#) | [Configuration](#) | [Modules](#) | [API](#) | [Misc](#)

Core Enhancements:

[Dynamic Shared Object \(DSO\) support](#)

Apache modules may now be loaded at runtime; this means that modules can be loaded into the server process space only when necessary, thus overall memory usage by Apache will be significantly reduced. DSO currently is supported on FreeBSD, OpenBSD, NetBSD, Linux, Solaris, SunOS, Digital UNIX, IRIX, HP/UX, UnixWare, NetWare, AIX, ReliantUnix and generic SVR4 platforms.

[Support for Windows NT/95](#)

Apache now supports the Windows NT and Windows 2000 operating systems. While Apache may run on Windows 95, 98, or ME, these consumer products are never recommended for production environments, and their use remains experimental. All versions of Apache running on Windows prior to 1.3.15 should be considered beta quality releases.

[Support for Cygwin](#)

Apache now supports the Cygwin platform for the Windows NT and Windows 2000 operating systems. The Cygwin versions should be considered as stable and reliable as the [Windows](#) native counterpart.

[Support for NetWare 5.x](#)

Apache now experimentally supports the NetWare 5.x operating systems.

[Re-organized Sources](#)

The source files for Apache have been re-organized. The main difference for Apache users is that the "Module" lines in Configuration have been replaced with "AddModule" with a slightly different syntax. For module authors there are some changes designed to make it easier for users to add their module.

Reliable Piped Logs

On almost all Unix architectures Apache now implements "reliable" piped logs in [mod_log_config](#). Where reliable means that if the logging child dies for whatever reason, Apache will recover and respawn it without having to restart the entire server. Furthermore if the logging child becomes "stuck" and isn't reading its pipe frequently enough Apache will also restart it. This opens up more opportunities for log rotation, hit filtering, real-time splitting of multiple vhosts into separate logs, and asynchronous DNS resolving on the fly.

Performance Improvements

- IP-based virtual hosts are looked up via hash table.
- <Directory> parsing speedups.
- The critical path for static requests has fewer system calls. This generally helps all requests. (45 syscalls for a static request in 1.2 versus 22 in 1.3 in a well tuned configuration).
- [ProxyReceiveBufferSize](#) directive gives mod_proxy's outgoing connections larger network buffers, for increased throughput.
- The low level I/O routines use `writew` (where available) to issue multiple writes with a single system call. They also avoid copying memory into buffers as much as possible. The result is less CPU time spent on transferring large files.
- Static requests are served using `mmap`, which means bytes are only copied from the disk buffer to the network buffer directly by the kernel. The program never copies bytes around, which reduces CPU time. (Only where available/tested.)
- When presented with a load spike, the server quickly adapts by spawning children at faster rates.
- The code which dispatches modules was optimized to avoid repeatedly skipping over modules that don't implement certain phases of the API. (This skipping showed up as 5% of the CPU time on profiles of a server with the default module mix.)
- Revamp of the Unix scoreboard management code so that less time is spent counting children in various states. Previously a scan was performed for each hit, now it is performed only once per second. This should be noticeable on servers running with hundreds of children and high loads.
- New serialization choices improve performance on Linux, and IRIX.
- [mod_log_config](#) can be compile-time configured to buffer writes.
- Replaced `strncpy()` with `ap_cpystrn()`, a routine which doesn't have to zero-fill the entire result. This has dramatic effects on `mod_include` speed.
- Additions to the internal "table" API (used for keeping lists of key/value string pairs) provide for up to 20% performance improvement in many situations.

See [the new performance documentation](#) for more information.

Configuration Enhancements

Unified Server Configuration Files

(*Apache 1.3.4*) The contents of the three server configuration files (`httpd.conf`, `srm.conf`, and `access.conf`) have been merged into a single `httpd.conf` file. The `srm.conf` and `access.conf` files are now empty except for comments directing the Webmaster to look in `httpd.conf`. In addition, the merged `httpd.conf` file has been restructured to allow directives to appear in a hopefully more intuitive and meaningful order.

Continuation Lines in config files

Directive lines in the server configuration files may now be split onto multiple lines by using the canonical Unix continuation mechanism, namely a `\` as the last non-blank character on the line to indicate that the next line should be concatenated.

Apache Autoconf-style Interface (APACI)

Until Apache 1.3 there was no real out-of-the-box batch-capable build and installation procedure for the complete Apache package. This is now provided by a top-level `configure` script and a corresponding top-level `Makefile.tmpl` file. The goal is to provide a GNU Autoconf-style frontend which is capable to both drive the old `src/Configure` stuff in batch and additionally installs the package with a GNU-conforming directory layout. Any options from the old configuration scheme are available plus a lot of new options for flexibly customizing Apache.

Note: The default installation layout has changed for Apache 1.3.4. See the files `README.configure` and `INSTALL` for more information.

Apache eXtenSion (APXS) support tool

Now that Apache provides full support for loading modules under runtime from dynamic shared object (DSO) files, a new support tool `apxs` was created which provides off-source building, installing and activating of those DSO-based modules. It completely hides the platform-dependent DSO-build commands from the user and provides an easy way to build modules

outside the Apache source tree. To achieve this APACI installs the Apache C header files together with the `apxs` tool.

[Default Apache directory path changed to /usr/local/apache/](#)

The default directory for the Apache ServerRoot changed from the NCSA-compatible `/usr/local/etc/httpd/` to `/usr/local/apache/`. This change covers only the default setting (and the documentation); it is of course possible to override it using the `-d ServerRoot` and `-f httpd.conf` switches when starting apache.

Improved HTTP/1.1-style Virtual Hosts

The new `NameVirtualHost` directive is used to list IP address:port pairs on which HTTP/1.1-style virtual hosting occurs. This is vhosting based on the `Host`: header from the client. Previously this address was implicitly the same as the "main address" of the machine, and this caused no end of problems for users, and was not powerful enough. Please see the [Apache Virtual Host documentation](#) for further details on configuration.

Include directive

The `Include` directive includes other config files immediately at that point in parsing.

-S command line option for debugging vhost setup

If Apache is invoked with the `-S` command line option it will dump out information regarding how it parsed the `VirtualHost` sections. This is useful for folks trying to debug their virtual host configuration.

Control of HTTP methods

`<LimitExcept>` and `</LimitExcept>` are used to enclose a group of access control directives which will then apply to any HTTP access method not listed in the arguments; i.e., it is the opposite of a `<Limit>` section and can be used to control both standard and nonstandard/unrecognized methods.

Module Enhancements

[Improved mod_negotiation](#)

The optional content negotiation (MultiViews) module has been completely overhauled for Apache 1.3.4, incorporating the latest HTTP/1.1 revisions and the experimental Transparent Content Negotiation features of RFC 2295 and RFC 2296.

[NEW - Spelling correction module](#)

This optional module corrects frequently occurring spelling and capitalization errors in document names requested from the server.

[NEW - Conditional setting of environment variables](#)

The addition of `SetEnvIf` and `SetEnvIfNoCase`. These allow you to set environment variables for server and CGI use based upon attributes of the request.

[NEW - "Magic" MIME-typing](#)

The optional `mod_mime_magic` has been added. It uses "magic numbers" and other hints from a file's contents to figure out what the contents are. It then uses this information to set the file's media type, if it cannot be determined by the file's extension.

[NEW - Unique Request Identifiers](#)

`mod_unique_id` can be included to generate a unique identifier that distinguishes a hit from every other hit. ("Unique" has some restrictions on it.) The identifier is available in the environment variable `UNIQUE_ID`.

mod_proxy enhancements:

- Easier and safer authentication for ftp proxy logins: When no ftp user name and/or password is specified in the URL, but the destination ftp server requires one, Apache now returns a "[401] Authorization Required" status. This status code usually makes the client browser pop up an "Enter user name and password" dialog, and the request is retried with the given user authentication. That is slightly more secure than specifying the authentication information as part of the request URL, where it could be logged in plaintext by older proxy servers.
- The new `AllowCONNECT` directive allows configuration of the port numbers to which the proxy `CONNECT` method may connect. That allows proxying to `https://some.server:8443/` which resulted in an error message prior to Apache version 1.3.2.
- The proxy now supports the HTTP/1.1 "Via:" header as specified in RFC2068. The new `ProxyVia` directive

allows switching "Via:" support off or on, or suppressing outgoing "Via:" header lines altogether for privacy reasons.

- The "Max-Forwards:" TRACE header specified in HTTP/1.1 is now supported. With it, you can trace the path of a request along a chain of proxies (if they, too, support it).
- [NoProxy](#) and [ProxyDomain](#) directives added to proxy, useful for intranets.
- New [ProxyPassReverse](#) directive. It lets Apache adjust the URL in the Location header on HTTP redirect responses.
- Easier navigation in ftp server directory trees.

[Enhanced mod_include string comparisons](#)

The string-based server-side include (SSI) flow-control directives now include comparison for less-than (<), less-than-or-equal (<=), greater-than (>), and greater-than-or-equal (>=). Previously comparisons could only be made for equality or inequality.

ServerRoot relative auth filenames

Auth filenames for the various authentication modules are now treated as relative to the ServerRoot if they are not full paths.

[Enhancements to directory indexing:](#)

- **Code split:** The mod_dir module has been split in two, with [mod_dir](#) handling directory index files, and [mod_autoindex](#) creating directory listings. Thus allowing folks to remove the indexing function from critical servers.
- **Sortable:** Clicking on a column title will now sort the listing in order by the values in that column. This feature can be disabled using the SuppressColumnSorting [IndexOptions](#) keyword.
- [SuppressHTMLPreamble](#) can be used if your README.html file includes its own HTML header.
- The [IndexOptions](#) directive now allows the use of incremental prefixes (+/- to add/remove the respective keyword feature, as was already possible for the [Options](#) directive) to its keyword arguments. Multiple IndexOptions directives applying to the same directory will now be merged.
- [IconHeight](#) and [IconWidth](#) let you set height and width attributes to the tag in directory listings.
- The new [NameWidth](#) keyword to the [IndexOptions](#) directive lets you set the number of columns for "[fancy](#)" [directory listings](#). If set to an '*' asterisk, the name width will be adjusted automatically.
- The [FancyIndexing](#) directive now correctly has the same impact as [IndexOptions FancyIndexing](#) without replacing the effect of any existing IndexOptions directive.
- Starting with 1.3.15, the server will satisfy directory requests with the cache controls ETag and LastModified, if IndexOptions includes the [TrackModified](#) directive. The server will not need to generate the listing if the client determines the request has not changed, improving performance. Due to its experimental nature, this feature is not enabled by default.

Less Buffering of CGI Script Output

In previous versions of Apache, the output from CGI scripts would be internally buffered by the server, and wouldn't be forwarded to the client until either the buffers were full or the CGI script completed. As of Apache 1.3, the buffer to the client is flushed any time it contains something and the server is waiting for more information from the script. This allows CGI script to provide partial status reports during long processing operations.

[Regular Expression support for Alias and Redirect](#)

New [AliasMatch](#), [ScriptAliasMatch](#), and [RedirectMatch](#) directives allow for the use of regular expression matching. Additionally, new [<DirectoryMatch>](#), [<LocationMatch>](#), and [<FilesMatch>](#) sections provide a new syntax for regular expression sectioning.

[AddModuleInfo](#) directive added to [mod_info](#)

Allows additional information to be listed along with a specified module.

Absence of any TransferLog disables logging

If no [TransferLog](#) directive is given then no log is written. This supports co-existence with other logging modules.

Ability to name logging formats

The [LogFormat](#) directive has been enhanced to allow you to give nicknames to specific logging formats. You can then use these nicknames in other [LogFormat](#) and [CustomLog](#) directives, rather than having to spell out the complete log format string each time.

Conditional logging

[mod_log_config](#) now supports logging based upon environment variables. [mod_log_referer](#) and [mod_log_agent](#) are now deprecated.

mod_cern_meta configurable per-directory

[mod_cern_meta](#) is now configurable on a per-directory basis.

New map types for [RewriteMap](#) directive

The new map types 'Randomized Plain Text' and 'Internal Function' were added to the [RewriteMap](#) directive of [mod_rewrite](#). They provide two new features: First, you now can randomly choose a sub-value from a value which was looked-up in a rewriting map (which is useful when choosing between backend servers in a Reverse Proxy situation). Second, you now can translate URL parts to fixed (upper or lower) case (which is useful when doing mass virtual hosting by the help of [mod_rewrite](#)).

CIDR and Netmask access control

[mod_access](#) directives now support CIDR (Classless Inter-Domain Routing) style prefixes, and netmasks for greater control over IP access lists.

API Additions and Changes

For all those module writers and code hackers:

child_init

A new phase for Apache's API is called once per "heavy-weight process," before any requests are handled. This allows the module to set up anything that need to be done once per processes. For example, connections to databases.

child_exit

A new phase called once per "heavy-weight process," when it is terminating. Note that it can't be called in some fatal cases (such as segfaults and kill -9). The `child_init` and `child_exit` functions are passed a pool whose lifetime is the same as the lifetime of the child (modulo completely fatal events in which Apache has no hope of recovering). In contrast, the module `init` function is passed a pool whose lifetime ends when the parent exits or restarts.

child_terminate

Used in the child to indicate the child should exit after finishing the current request.

register_other_child

See `http_main.h`. This is used in the parent to register a child for monitoring. The parent will report status to a supplied callback function. This allows modules to create their own children which are monitored along with the `httpd` children.

piped_log

See `http_log.h`. This API provides the common code for implementing piped logs. In particular it implements a reliable piped log on architectures supporting it (*i.e.*, Unix at the moment).

scoreboard format changed

The scoreboard format is quite different. It is considered a "private" interface in general, so it's only mentioned here as an FYI.

set_last_modified split into three

The old function `set_last_modified` performed multiple jobs including the setting of the `Last-Modified` header, the `ETag` header, and processing conditional requests (such as IMS). These functions have been split into three functions: `set_last_modified`, `set_etag`, and `meets_conditions`. The field `mtime` has been added to `request_rec` to facilitate `meets_conditions`.

New error logging function: `ap_log_error`

All old logging functions are deprecated, we are in the process of replacing them with a single function called `ap_log_error`. This is still a work in progress.

set_file_slot for config parsing

The `set_file_slot` routine provides a standard routine that prepends `ServerRoot` to non-absolute paths.

post_read_request module API

This request phase occurs immediately after reading the request (headers), and immediately after creating an internal redirect. It is most useful for setting environment variables to affect future phases.

psocket, and popendir

The `psocket` and `pclosesocket` functions allow for race-condition free socket creation with resource tracking. Similarly `popendir` and `pclosedir` protect directory reading.

is_initial_req

Test if the request is the initial request (*i.e.*, the one coming from the client).

kill_only_once

An option to `ap_spawn_child` functions which prevents Apache from aggressively trying to kill off the child.

alloc debugging code

Defining `ALLOC_DEBUG` provides a rudimentary memory debugger which can be used on live servers with low impact -- it sets all allocated and freed memory bytes to `0xa5`. Defining `ALLOC_USE_MALLOC` will cause the alloc code to use `malloc()` and `free()` for each object. This is far more expensive and should only be used for testing with tools such as Electric Fence and Purify. See `main/alloc.c` for more details.

ap_cpystn

The new `strncpy` "lookalike", with slightly different semantics is much faster than `strncpy` because it doesn't have to zero-fill the entire buffer.

table_addn, table_setn, table_mergen

These new functions do **not** call `pstrdup` on their arguments. This provides for big speedups. There is also some debugging support to ensure code uses them properly. See `src/CHANGES` for more information.

construct_url

The function prototype for this changed from taking a `server_rec *` to taking a `request_rec *`.

get_server_name, get_server_port

These are wrappers which deal with the [UseCanonicalName](#) directive when retrieving the server name and port for a request.

Change to prototype for ap_bspawn_child and ap_call_exec

Added a `child_info *` to spawn function (as passed to `ap_bspawn_child`) and to `ap_call_exec` to allow children to work correctly on Win32. We also cleaned up the nomenclature a bit, replacing `spawn_child_err` with simply `ap_spawn_child` and `spawn_child_err_buff` with simply `ap_bspawn_child`.

ap_add_version_component()

This API function allows for modules to add their own additional server tokens which are printed on the on the `Server:` header line. Previous 1.3beta versions had used a `SERVER_SUBVERSION` compile-time `#define` to perform this function. Whether the tokens are actually displayed is controlled by the new `ServerTokens` directive.

Miscellaneous Enhancements

[Port to EBCDIC mainframe machine running BS2000/OSD](#)

As a premiere, this version of Apache comes with a beta version of a port to a mainframe machine which uses the EBCDIC character set as its native codeset (It is the SIEMENS family of mainframes running the BS2000/OSD operating system on a IBM/390 compatible processor. This mainframe OS nowadays features a SVR4-like POSIX subsystem).

[AccessFileName Enhancement](#)

The `AccessFileName` directive can now take more than one filename. This lets sites serving pages from network file systems and more than one Apache web server, configure access based on the server through which shared pages are being served.

HostNameLookups now defaults to "Off"

The [HostNameLookups](#) directive now defaults to "Off". This means that, unless explicitly turned on, the server will not resolve IP addresses into names. This was done to spare the Internet from unnecessary DNS traffic.

Double-Reverse DNS enforced

The [HostnameLookups](#) directive now supports double-reverse DNS. (Known as *PARANOID* in the terminology of `tcp_wrappers`.) An IP address passes a double-reverse DNS test if the forward map of the reverse map includes the original IP. Regardless of the `HostnameLookups` setting, [mod_access](#) access lists using DNS names **require** all names to pass a double-reverse DNS test. (Prior versions of Apache required a compile-time switch to enable double-reverse DNS.)

LogLevel and syslog support

Apache now has [configurable error logging levels](#) and supports [error logging via syslogd\(8\)](#).

Detaching from stdin/out/err

On boot Apache will now detach from `stdin`, `stdout`, and `stderr`. It does not detach from `stderr` until it has successfully read the config files. So you will see errors in the config file. This should make it easier to start Apache via `rsh` or `crontab`.

Year-2000 Improvements

The default `timefmt` string used by [mod_include](#) has been modified to display the year using four digits rather than the two-digit format used previously. The [mod_autoindex](#) module has also been modified to display years using four digits in FancyIndexed directory listings.

Common routines Moving to a Separate Library

There are a number of functions and routines that have been developed for the Apache project that supplement or supersede library routines that differ from one operating system to another. While most of these are used only by the Apache server itself, some are referenced by supporting applications (such as `htdigest`), and these other applications would fail to build because the routines were built only into the server. These routines are now being migrated to a separate subdirectory and library so they can be used by other applications than just the server. See the `src/ap/` subdirectory.

New [ServerSignature](#) directive

This directive optionally adds a line containing the server version and virtual host name to server-generated pages (error documents, ftp directory listings, `mod_info` output *etc.*). This makes it easier for users to tell which server produced the error message, especially in a proxy chain (often found in intranet environments).

New [UseCanonicalName](#) directive

This directive gives control over how Apache creates self-referential URLs. Previously Apache would always use the [ServerName](#) and [Port](#) directives to construct a "canonical" name for the server. With `UseCanonicalName off` Apache will use the hostname and port supplied by the client, if available.

SERVER_VERSION definition abstracted, and server build date added

In earlier versions, the Apache server version was available to modules through the `#defined` value for `SERVER_VERSION`. In order to keep this value consistent when modules and the core server are compiled at different times, this information is now available through the core API routine `ap_get_server_version()`. The use of the `SERVER_VERSION` symbol is deprecated. Also, `ap_get_server_built()` returns a string representing the time the core server was linked.

[Including the operating system in the server identity](#)

A new directive, `ServerTokens`, allows the Webmaster to change the value of the `Server` response header field which is sent back to clients. The `ServerTokens` directive controls whether the server will include a non-specific note in the server identity about the type of operating system on which the server is running as well as included module information. As of Apache 1.3, this additional information is included by default.

Support for Netscape style SHA1 encrypted passwords

To facilitate migration or integration of BasicAuth password schemes where the password is encrypted using SHA1 (as opposed to Apache's built in MD5 and/or the OS specific `crypt(3)` function) passwords prefixed with `{SHA1}` are taken as Base64 encoded SHA1 passwords. More information and some utilities to convert Netscape `ldap/ldif` entries can be found in `support/SHA1`.

Apache HTTP Server





Apache HTTP Server

Compiling and Installing Apache 1.3

This document covers compilation and installation of Apache on Unix systems, using the manual build and install method. If you wish to use the autoconf-style configure interface, you should instead read the `INSTALL` file in the root directory of the Apache source distribution. For compiling and installation on specific platforms, see

- [Using Apache with Microsoft Windows](#)
- [Using Apache with Cygwin](#)
- [Using Apache with Novell Netware 5](#)
- [Using Apache with HP MPE/iX](#)
- [Compiling Apache under UnixWare](#)
- [Overview of the Apache TPF Port](#)

Downloading Apache

Information on the latest version of Apache can be found on the Apache web server at <http://www.apache.org/>. This will list the current release, any more recent beta-test release, together with details of mirror web and anonymous ftp sites.

If you downloaded a binary distribution, skip to [Installing Apache](#). Otherwise read the next section for how to compile the server.

Compiling Apache

Compiling Apache consists of three steps: Firstly select which Apache **modules** you want to include into the server. Secondly create a configuration for your operating system. Thirdly compile the executable.

All configuration of Apache is performed in the `src` directory of the Apache distribution. Change into this directory.

1. Select modules to compile into Apache in the `Configuration` file. Uncomment lines corresponding to those optional modules you wish to include (among the `AddModule` lines at the bottom of the file), or add new lines corresponding to additional modules you have downloaded or written. (See [API.html](#) for preliminary docs on how to write Apache modules). Advanced users can comment out some of the default modules if they are sure they will not need them (be careful though, since many of the default modules are vital for the correct operation and security of the server).

You should also read the instructions in the `Configuration` file to see if you need to set any of the `Rule` lines.

2. Configure Apache for your operating system. Normally you can just run the `Configure` script as given below. However if this fails or you have any special requirements (e.g., to include an additional library required by an optional module) you might need to edit one or more of the following options in the `Configuration` file: `EXTRA_CFLAGS`, `LIBS`, `LDLFLAGS`, `INCLUDES`.

Run the `Configure` script:

```
% Configure
Using 'Configuration' as config file
+ configured for <whatever> platform
+ setting C compiler to <whatever> *
+ setting C compiler optimization-level to <whatever> *
```

```
+ Adding selected modules
+ doing sanity check on compiler and options
Creating Makefile in support
Creating Makefile in main
Creating Makefile in os/unix
Creating Makefile in modules/standard
```

(*: Depending on Configuration and your system, Configure might not print these lines. That's OK).

This generates a Makefile for use in stage 3. It also creates a Makefile in the support directory, for compilation of the optional support programs.

(If you want to maintain multiple configurations, you can give an option to Configure to tell it to read an alternative Configuration file, such as `Configure -file Configuration.ai`).

3. Type `make`.

The modules we place in the Apache distribution are the ones we have tested and are used regularly by various members of the Apache development group. Additional modules contributed by members or third parties with specific needs or functions are available at <http://www.apache.org/dist/httpd/contrib/modules/>. There are instructions on that page for linking these modules into the core Apache code.

Installing Apache

You will have a binary file called `httpd` in the `src` directory. A binary distribution of Apache will supply this file.

The next step is to install the program and configure it. Apache is designed to be configured and run from the same set of directories where it is compiled. If you want to run it from somewhere else, make a directory and copy the `conf`, `logs` and `icons` directories into it. In either case you should read the [security tips](#) describing how to set the permissions on the server root directory.

The next step is to edit the configuration files for the server. This consists of setting up various **directives** in up to three central configuration files. By default, these files are located in the `conf` directory and are called `srm.conf`, `access.conf` and `httpd.conf`. To help you get started there are same files in the `conf` directory of the distribution, called `srm.conf-dist`, `access.conf-dist` and `httpd.conf-dist`. Copy or rename these files to the names without the `-dist`. Then edit each of the files. Read the comments in each file carefully. Failure to setup these files correctly could lead to your server not working or being insecure. You should also have an additional file in the `conf` directory called `mime.types`. This file usually does not need editing.

First edit `httpd.conf`. This sets up general attributes about the server: the port number, the user it runs as, *etc.* Next edit the `srm.conf` file; this sets up the root of the document tree, special functions like server-parsed HTML or internal imagemap parsing, *etc.* Finally, edit the `access.conf` file to at least set the base cases of access.

In addition to these three files, the server behavior can be configured on a directory-by-directory basis by using `.htaccess` files in directories accessed by the server.

Set your system time properly!

Proper operation of a public web server requires accurate time keeping, since elements of the HTTP protocol are expressed as the time of day. So, it's time to investigate setting up NTP or some other time synchronization system on your Unix box, or whatever the equivalent on NT would be.

Compiling Support Programs

In addition to the main `httpd` server which is compiled and configured as above, Apache includes a number of support programs. These are not compiled by default. The support programs are in the `support` directory of the distribution. To compile the support programs, change into this directory and type

```
make
```

Apache HTTP Server





Apache HTTP Server

Starting Apache

- [Starting Apache on Windows](#)
 - [Starting Apache on Unix](#)
 - [Errors During Start-up](#)
 - [Starting at Boot-Time](#)
 - [Additional Information](#)
-

Starting Apache On Windows

On Windows, Apache is normally run as a service on Windows NT, or as a console application on Windows 95. This does not apply in its full extent for the Cygwin platform. For details, see [running Apache for Windows](#).

Starting Apache on Unix

On Unix, the [httpd](#) program is run as a daemon which executes continuously in the background to handle requests. It is possible to have Apache invoked by the Internet daemon `inetd` each time a connection to the HTTP service is made using the [ServerType](#) directive, but this is not recommended.

If the [Port](#) specified in the configuration file is the default of 80 (or any other port below 1024), then it is necessary to have root privileges in order to start Apache, so that it can bind to this privileged port. Once the server has started and completed a few preliminary activities such as opening its log files, it will launch several *child* processes which do the work of listening for and answering requests from clients. The main `httpd` process continues to run as the root user, but the child processes run as a less privileged user. This is controlled by Apache's [process creation directives](#).

The first thing that `httpd` does when it is invoked is to locate and read the [configuration file](#) `httpd.conf`. The location of this file is set at compile-time, but it is possible to specify its location at run time using the `-f` command-line option as in

```
/usr/local/apache/bin/httpd -f /usr/local/apache/conf/httpd.conf
```

As an alternative to invoking the `httpd` binary directly, a shell script called [apachectl](#) is provided which can be used to control the daemon process with simple commands such as `apachectl start` and `apachectl stop`.

If all goes well during startup, the server will detach from the terminal and the command prompt will return almost immediately. This indicates that the server is up and running. You can then use your browser to connect to the server and view the test page in the [DocumentRoot](#) directory and the local copy of the documentation linked from that page.

Errors During Start-up

If Apache suffers a fatal problem during startup, it will write a message describing the problem either to the console or to the [ErrorLog](#) before exiting. One of the most common error messages is "Unable to bind to Port ...". This message is usually caused by either:

- Trying to start the server on a privileged port when not logged in as the root user; or

Starting Apache

- Trying to start the server when there is another instance of Apache or some other web server already bound to the same port.

For further trouble-shooting instructions, consult the Apache [FAQ](#).

Starting at Boot-Time

If you want your server to continue running after a system reboot, you should add a call to `httpd` or `apachectl` to your system startup files (typically `rc.local` or a file in an `rc.N` directory). This will start Apache as root. Before doing this ensure that your server is properly configured for security and access restrictions. The `apachectl` script is designed so that it can often be linked directly as an init script, but be sure to check the exact requirements of your system.

Additional Information

Additional information about the command-line options of [httpd](#) and [apachectl](#) as well as other support programs included with the server is available on the [Server and Supporting Programs](#) page. There is also documentation on all the [modules](#) included with the Apache distribution and the [directives](#) that they provide.

Apache HTTP Server





Apache HTTP Server

Stopping and Restarting Apache

This document covers stopping and restarting Apache on Unix and Cygwin only. Windows users should see [Signalling Apache when running](#).

You will notice many `httpd` executables running on your system, but you should not send signals to any of them except the parent, whose pid is in the [PidFile](#). That is to say you shouldn't ever need to send signals to any process except the parent. There are three signals that you can send the parent: `TERM`, `HUP`, and `USR1`, which will be described in a moment.

To send a signal to the parent you should issue a command such as:

```
kill -TERM `cat /usr/local/apache/logs/httpd.pid`
```

You can read about its progress by issuing:

```
tail -f /usr/local/apache/logs/error_log
```

Modify those examples to match your [ServerRoot](#) and [PidFile](#) settings.

As of Apache 1.3 we provide a script called [apachectl](#) which can be used to start, stop, and restart Apache. It may need a little customization for your system, see the comments at the top of the script.

TERM Signal: stop now

Sending the `TERM` signal to the parent causes it to immediately attempt to kill off all of its children. It may take it several seconds to complete killing off its children. Then the parent itself exits. Any requests in progress are terminated, and no further requests are served.

HUP Signal: restart now

Sending the `HUP` signal to the parent causes it to kill off its children like in `TERM` but the parent doesn't exit. It re-reads its configuration files, and re-opens any log files. Then it spawns a new set of children and continues serving hits.

Users of the [status module](#) will notice that the server statistics are set to zero when a `HUP` is sent.

Note: If your configuration file has errors in it when you issue a restart then your parent will not restart, it will exit with an error. See below for a method of avoiding this.

USR1 Signal: graceful restart

Note: prior to release 1.2b9 this code is quite unstable and shouldn't be used at all.

The `USR1` signal causes the parent process to *advise* the children to exit after their current request (or to exit immediately if they're not serving anything). The parent re-reads its configuration files and re-opens its log files. As each child dies off the parent replaces it with a child from the new *generation* of the configuration, which begins serving new requests immediately.

This code is designed to always respect the [MaxClients](#), [MinSpareServers](#), and [MaxSpareServers](#) settings. Furthermore, it respects [StartServers](#) in the following manner: if after one second at least `StartServers` new children have not been created, then create enough to pick up the slack. This is to say that the code tries to maintain both the number of children appropriate for the current load

on the server, and respect your wishes with the `StartServers` parameter.

Users of the [status module](#) will notice that the server statistics are **not** set to zero when a `USR1` is sent. The code was written to both minimize the time in which the server is unable to serve new requests (they will be queued up by the operating system, so they're not lost in any event) and to respect your tuning parameters. In order to do this it has to keep the *scoreboard* used to keep track of all children across generations.

The status module will also use a `G` to indicate those children which are still serving requests started before the graceful restart was given.

At present there is no way for a log rotation script using `USR1` to know for certain that all children writing the pre-restart log have finished. We suggest that you use a suitable delay after sending the `USR1` signal before you do anything with the old log. For example if most of your hits take less than 10 minutes to complete for users on low bandwidth links then you could wait 15 minutes before doing anything with the old log.

Note: If your configuration file has errors in it when you issue a restart then your parent will not restart, it will exit with an error. In the case of graceful restarts it will also leave children running when it exits. (These are the children which are "gracefully exiting" by handling their last request.) This will cause problems if you attempt to restart the server -- it will not be able to bind to its listening ports. Before doing a restart, you can check the syntax of the configuration files with the `-t` command line argument (see [httpd](#)). This still will not guarantee that the server will restart correctly. To check the semantics of the configuration files as well as the syntax, you can try starting `httpd` as a non-root user. If there are no errors it will attempt to open its sockets and logs and fail because it's not root (or because the currently running `httpd` already has those ports bound). If it fails for any other reason then it's probably a config file error and the error should be fixed before issuing the graceful restart.

Appendix: signals and race conditions

Prior to Apache 1.2b9 there were several *race conditions* involving the restart and die signals (a simple description of race condition is: a time-sensitive problem, as in if something happens at just the wrong time it won't behave as expected). For those architectures that have the "right" feature set we have eliminated as many as we can. But it should be noted that there still do exist race conditions on certain architectures.

Architectures that use an on disk [ScoreBoardFile](#) have the potential to corrupt their scoreboards. This can result in the "bind: Address already in use" (after `HUP`) or "long lost child came home!" (after `USR1`). The former is a fatal error, while the latter just causes the server to lose a scoreboard slot. So it might be advisable to use graceful restarts, with an occasional hard restart. These problems are very difficult to work around, but fortunately most architectures do not require a scoreboard file. See the [ScoreBoardFile](#) documentation for a architecture uses it.

`NEXT` and `MACHTEN` (68k only) have small race conditions which can cause a restart/die signal to be lost, but should not cause the server to do anything otherwise problematic.

All architectures have a small race condition in each child involving the second and subsequent requests on a persistent HTTP connection (`KeepAlive`). It may exit after reading the request line but before reading any of the request headers. There is a fix that was discovered too late to make 1.2. In theory this isn't an issue because the `KeepAlive` client has to expect these events because of network latencies and server timeouts. In practice it doesn't seem to affect anything either -- in a test case the server was restarted twenty times per second and clients successfully browsed the site without getting broken images or empty documents.

Apache HTTP Server





Apache HTTP Server

Configuration Files

- [Main Configuration Files](#)
- [Syntax of the Configuration Files](#)
- [Modules](#)
- [Scope of Directives](#)
- [.htaccess Files](#)

Main Configuration Files

Related Modules	Related Directives
mod_mime	AccessConfig <IfDefine> Include ResourceConfig TypesConfig

Apache is configured by placing [directives](#) in plain text configuration files. The main configuration file is usually called `httpd.conf`. The location of this file is set at compile-time, but may be overridden with the `-f` command line flag. Some sites also have `srm.conf` and `access.conf` files for [historical reasons](#). In addition, other configuration files may be added using the [Include](#) directive. Any directive may be placed in any of these configuration files. Changes to the main configuration files are only recognized by Apache when it is started or restarted.

New with Apache 1.3.13 is a feature where if any configuration file is actually a directory, Apache will enter that directory and parse any files (and subdirectories) found there as configuration files. One possible use for this would be to add VirtualHosts by creating small configuration files for each host, and placing them in such a configuration directory. Thus, you can add or remove VirtualHosts without editing any files at all, simply adding or deleting them. This makes automating such processes much easier.

The server also reads a file containing mime document types; the filename is set by the [TypesConfig](#) directive, and is `mime.types` by default.

Syntax of the Configuration Files

Apache configuration files contain one directive per line. The back-slash `"\"` may be used as the last character on a line to indicate that the directive continues onto the next line. There must be no other characters or white space between the back-slash and the end of the line.

Directives in the configuration files are case-insensitive, but arguments to directives are often case sensitive. Lines which begin with the hash character `"#"` are considered comments, and are ignored. Comments may **not** be included on a line after a configuration directive. Blank lines and white space occurring before a directive are ignored, so you may indent directives for clarity.

You can check your configuration files for syntax errors without starting the server by using `apachectl configtest` or the `-t` command line option.

Modules

Related Modules	Related Directives
mod_so	AddModule ClearModuleList <IfModule> LoadModule

Apache is a modular server. This implies that only the most basic functionality is included in the core server. Extended features are available through [modules](#) which can be loaded into Apache. By default, a [base](#) set of modules is included in the server at compile-time. If the server is compiled to use [dynamically loaded](#) modules, then modules can be compiled separately and added at any time using the [LoadModule](#) directive. Otherwise, Apache must be recompiled to add or remove modules. Configuration directives may be included conditional on a presence of a particular module by enclosing them in an [<IfModule>](#) block.

To see which modules are currently compiled into the server, you can use the `-l` command line option.

Scope of Directives

Related Directives
<Directory> <DirectoryMatch> <Files> <FilesMatch> <Location> <LocationMatch> <VirtualHost>

Directives placed in the main configuration files apply to the entire server. If you wish to change the configuration for only a part of the server, you can scope your directives by placing them in [<Directory>](#), [<DirectoryMatch>](#), [<Files>](#), [<FilesMatch>](#), [<Location>](#), and [<LocationMatch>](#) sections. These sections limit the application of the directives which they enclose to particular filesystem locations or URLs. They can also be nested, allowing for very fine grained configuration.

Apache has the capability to serve many different websites simultaneously. This is called [Virtual Hosting](#). Directives can also be scoped by placing them inside [<VirtualHost>](#) sections, so that they will only apply to requests for a particular website.

Although most directives can be placed in any of these sections, some directives do not make sense in some contexts. For example, directives controlling process creation can only be placed in the main server context. To find which directives can be placed in which sections, check the [Context](#) of the directive. For further information, we provide details on [How Directory, Location and Files sections work](#).

.htaccess Files

Related Directives[AccessFileName](#)[AllowOverride](#)

Apache allows for decentralized management of configuration via special files placed inside the web tree. The special files are usually called `.htaccess`, but any name can be specified in the [AccessFileName](#) directive. Directives placed in `.htaccess` files apply to the directory where you place the file, and all sub-directories. The `.htaccess` files follow the same syntax as the main configuration files. Since `.htaccess` files are read on every request, changes made in these files take immediate effect.

To find which directives can be placed in `.htaccess` files, check the [Context](#) of the directive. The server administrator further controls what directives may be placed in `.htaccess` files by configuring the [AllowOverride](#) directive in the main configuration files.

For more information on `.htaccess` files, see Ken Coar's tutorial on [Using .htaccess Files with Apache](#).

Apache HTTP Server





Apache HTTP Server

How Directory, Location and Files sections work

The sections `<Directory>`, `<Location>` and `<Files>` can contain directives which only apply to specified directories, URLs or files respectively. Also `htaccess` files can be used inside a directory to apply directives to that directory. This document explains how these different sections differ and how they relate to each other when Apache decides which directives apply for a particular directory or request URL.

Directives allowed in the sections

Everything that is syntactically allowed in `<Directory>` is also allowed in `<Location>` (except a sub-`<Files>` section). Semantically, however some things, most notably `AllowOverride` and the two options `FollowSymLinks` and `SymLinksIfOwnerMatch`, make no sense in `<Location>`, `<LocationMatch>` or `<DirectoryMatch>`. The same for `<Files>` -- syntactically everything is fine, but semantically some things are different.

How the sections are merged

The order of merging is:

1. `<Directory>` (except regular expressions) and `.htaccess` done simultaneously (with `.htaccess`, if allowed, overriding `<Directory>`)
2. `<DirectoryMatch>`, and `<Directory>` with regular expressions
3. `<Files>` and `<FilesMatch>` done simultaneously
4. `<Location>` and `<LocationMatch>` done simultaneously

Apart from `<Directory>`, each group is processed in the order that they appear in the configuration files. `<Directory>` (group 1 above) is processed in the order shortest directory component to longest. If multiple `<Directory>` sections apply to the same directory they they are processed in the configuration file order. The configuration files are read in the order `httpd.conf`, `srm.conf` and `access.conf`. Configurations included via the `Include` directive will be treated as if they were inside the including file at the location of the `Include` directive.

Sections inside `<VirtualHost>` sections are applied *after* the corresponding sections outside the virtual host definition. This allows virtual hosts to override the main server configuration. (Note: this only works correctly from 1.2.2 and 1.3a2 onwards. Before those releases sections inside virtual hosts were applied *before* the main server).

Later sections override earlier ones.

Notes about using sections

The general guidelines are:

- If you are attempting to match objects at the filesystem level then you must use `<Directory>` and/or `<Files>`.
- If you are attempting to match objects at the URL level then you must use `<Location>`

But a notable exception is:

- proxy control is done via `<Directory>`. This is a legacy mistake because the proxy existed prior to `<Location>`. A future version of the config language should probably switch this to `<Location>`.

Note about .htaccess parsing:

- Modifying .htaccess parsing during Location doesn't do anything because .htaccess parsing has already occurred.

<Location> and symbolic links:

- It is not possible to use "Options FollowSymLinks" or "Options SymLinksIfOwnerMatch" inside a <Location>, <LocationMatch> or <DirectoryMatch> section (the options are simply ignored). Using the options in question is only possible inside a <Directory> section (or a .htaccess file).

<Files> and Options:

- Apache won't check for it, but using an Options directive inside a <Files> section has no effect.

Another note:

- There is actually a <Location>/<LocationMatch> sequence performed just before the name translation phase (where Aliases and DocumentRoots are used to map URLs to filenames). The results of this sequence are completely thrown away after the translation has completed.

Apache HTTP Server





Apache HTTP Server

Server-Wide Configuration

This document explains some of the directives provided by the [core](#) server which are used to configure the basic operations of the server.

- [Server Identification](#)
 - [File Locations](#)
 - [Process Creation](#)
 - [Network Configuration](#)
 - [Limiting Resource Usage](#)
-

Server Identification

Related Directives

[ServerName](#)
[ServerAdmin](#)
[ServerSignature](#)
[ServerTokens](#)
[UseCanonicalName](#)

The `ServerAdmin` and `ServerTokens` directives control what information about the server will be presented in server-generated documents such as error messages. The `ServerTokens` directive sets the value of the Server HTTP response header field.

The `ServerName` and `UseCanonicalName` directives are used by the server to determine how to construct self-referential URLs. For example, when a client requests a directory, but does not include the trailing slash in the directory name, Apache must redirect the client to the full name including the trailing slash so that the client will correctly resolve relative references in the document.

File Locations

Related Directives

[CoreDumpDirectory](#)
[DocumentRoot](#)
[ErrorLog](#)
[Lockfile](#)
[PidFile](#)
[ScoreBoardFile](#)
[ServerRoot](#)

These directives control the locations of the various files that Apache needs for proper operation. When the pathname used does not begin with a slash "/", the files are located relative to the `ServerRoot`. Be careful about locating files in paths which are writable by non-root users. See the [security tips](#) documentation for more details.

Process Creation

Related Directives

[BS2000Account](#)
[Group](#)
[MaxClients](#)
[MaxRequestsPerChild](#)
[MaxSpareServers](#)
[MinSpareServers](#)
[ServerType](#)
[StartServers](#)
[ThreadsPerChild](#)
[User](#)

When `ServerType` is set to its recommended value of `Standalone`, Apache 1.3 for Unix is a pre-forking web server. A single control process is responsible for launching child processes which listen for connections and serve them when they arrive. Apache always tries to maintain several *spare* or idle server processes, which stand ready to serve incoming requests. In this way, clients do not need to wait for a new child processes to be forked before their requests can be served.

The `StartServers`, `MinSpareServers`, `MaxSpareServers`, and `MaxServers` regulate how the parent process creates children to serve requests. In general, Apache is very self-regulating, so most sites do not need to adjust these directives from their default values. Sites which need to serve more than 256 simultaneous requests may need to increase `MaxClients`, while sites with limited memory may need to decrease `MaxClients` to keep the server from thrashing (swapping memory to disk and back). More information about tuning process creation is provided in the [performance hints](#) documentation.

While the parent process is usually started as root under Unix in order to bind to port 80, the child processes are launched by Apache as a less-privileged user. The `User` and `Group` directives are used to set the privileges of the Apache child processes. The child processes must be able to read all the content that will be served, but should have as few privileges beyond that as possible. In addition, unless `suexec` is used, these directives also set the privileges which will be inherited by CGI scripts.

`MaxRequestsPerChild` controls how frequently the server recycles processes by killing old ones and launching new ones.

Under Windows, Apache launches one control process and one child process. The child process creates multiple threads to serve requests. The number of threads is controlled by the `ThreadsPerChild` directive.

Network Configuration

Related Directives

[BindAddress](#)
[KeepAlive](#)
[KeepAliveTimeout](#)
[Listen](#)
[ListenBackLog](#)
[AcceptFilter](#)
[AcceptMutex](#)
[MaxKeepAliveRequests](#)
[Port](#)
[SendBufferSize](#)
[TimeOut](#)

When Apache starts, it connects to some port and address on the local machine and waits for incoming requests. By default, it listens to all addresses on the machine, and to the port as specified by the `Port` directive in the server configuration. However, it can be told to listen to more than one port, to listen to only selected addresses, or a combination. This is often combined with the [Virtual Host](#) feature which determines how Apache responds to different IP addresses, hostnames and ports.

There are two directives used to restrict or specify which addresses and ports Apache listens to. The `BindAddress` directive is used to restrict the server to listening to a single IP address. The `Listen` directive can be used to specify multiple IP addresses and/or Ports to which Apache will listen.

The `ListenBackLog`, `SendBufferSize`, and `TimeOut` directives are used to adjust how Apache interacts with the network. `AcceptFilter` controls a BSD specific filter optimization. See the BSD section on [performance hints](#) documentation. `AcceptMutex` controls which accept mutex method will be used. For an explanation of what this is and why it's needed, see the [performance tuning guide](#)

The `KeepAlive`, `KeepAliveTimeout`, and `MaxKeepAliveRequests` directives are used to configure how Apache handles persistent connections.

Limiting Resource Usage

Related Directives

[LimitRequestBody](#)
[LimitRequestFields](#)
[LimitRequestFieldsize](#)
[LimitRequestLine](#)
[RLimitCPU](#)
[RLimitMEM](#)
[RLimitNPROC](#)
[ThreadStackSize](#)

The `LimitRequest*` directives are used to place limits on the amount of resources Apache will use in reading requests from clients. By limiting these values, some kinds of denial of service attacks can be mitigated.

The `RLimit*` directives are used to limit the amount of resources which can be used by processes forked off from the Apache children. In particular, this will control resources used by CGI scripts and SSI exec commands.

The `ThreadStackSize` directive is used only on Netware to control the stack size.

Apache HTTP Server





Apache HTTP Server

Log Files

In order to effectively manage a web server, it is necessary to get feedback about the activity and performance of the server as well as any problems that may be occurring. The Apache HTTP Server provides very comprehensive and flexible logging capabilities. This document describes how to configure its logging capabilities, and how to understand what the logs contain.

- [Security Warning](#)
- [Error Log](#)
- [Access Log](#)
 - [Common Log Format](#)
 - [Combined Log Format](#)
 - [Multiple Access Logs](#)
 - [Conditional Logging](#)
- [Log Rotation](#)
- [Piped Logs](#)
- [VirtualHosts](#)
- [Other Log Files](#)
 - [PID File](#)
 - [Script Log](#)
 - [Rewrite Log](#)

Security Warning

Anyone who can write to the directory where Apache is writing a log file can almost certainly gain access to the uid that the server is started as, which is normally root. Do *NOT* give people write access to the directory the logs are stored in without being aware of the consequences; see the [security tips](#) document for details.

In addition, log files may contain information supplied directly by the client, without escaping. Therefore, it is possible for malicious clients to insert control-characters in the log files, so care must be taken in dealing with raw logs.

Error Log

Related Directives

[ErrorLog](#)
[LogLevel](#)

The server error log, whose name and location is set by the [ErrorLog](#) directive, is the most important log file. This is the place

where Apache httpd will send diagnostic information and record any errors that it encounters in processing requests. It is the first place to look when a problem occurs with starting the server or with the operation of the server, since it will often contain details of what went wrong and how to fix it.

The error log is usually written to a file (typically `error_log` on unix systems and `error.log` on Windows and OS/2). On unix systems it is also possible to have the server send errors to `syslog` or [pipe them to a program](#).

The format of the error log is relatively free-form and descriptive. But there is certain information that is contained in most error log entries. For example, here is a typical message.

```
[Wed Oct 11 14:32:52 2000] [error] [client 127.0.0.1] client denied by server configuration: /export/home/live/ap/htdocs/test
```

The first item in the log entry is the date and time of the message. The second entry lists the severity of the error being reported. The [LogLevel](#) directive is used to control the types of errors that are sent to the error log by restricting the severity level. The third entry gives the IP address of the client that generated the error. Beyond that is the message itself, which in this case indicates that the server has been configured to deny the client access. The server reports the file-system path (as opposed to the web path) of the requested document.

A very wide variety of different messages can appear in the error log. Most look similar to the example above. The error log will also contain debugging output from CGI scripts. Any information written to `stderr` by a CGI script will be copied directly to the error log.

It is not possible to customize the error log by adding or removing information. However, error log entries dealing with particular requests have corresponding entries in the [access log](#). For example, the above example entry corresponds to an access log entry with status code 403. Since it is possible to customize the access log, you can obtain more information about error conditions using that log file.

During testing, it is often useful to continuously monitor the error log for any problems. On unix systems, you can accomplish this using:

```
tail -f error_log
```

Access Log

Related Modules	Related Directives
mod_log_config	CustomLog LogFormat SetEnvIf

The server access log records all requests processed by the server. The location and content of the access log are controlled by the [CustomLog](#) directive. The [LogFormat](#) directive can be used to simplify the selection of the contents of the logs. This section describes how to configure the server to record information in the access log.

Of course, storing the information in the access log is only the start of log management. The next step is to analyze this information to produce useful statistics. Log analysis in general is beyond the scope of this document, and not really part of the job of the web server itself. For more information about this topic, and for applications which perform log analysis, check the [Open Directory](#) or [Yahoo](#).

Various versions of Apache httpd have used other modules and directives to control access logging, including `mod_log_referer`, `mod_log_agent`, and the `TransferLog` directive. The `CustomLog` directive now subsumes the functionality of all the older directives.

The format of the access log is highly configurable. The format is specified using a [format string](#) that looks much like a C-style `printf(1)` format string. Some examples are presented in the next sections. For a complete list of the possible contents of the format string, see the [mod_log_config documentation](#).

Common Log Format

A typical configuration for the access log might look as follows.

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
```

This defines the *nickname* `common` and associates it with a particular log format string. The format string consists of percent directives, each of which tell the server to log a particular piece of information. Literal characters may also be placed in the format string and will be copied directly into the log output. The quote character (") must be escaped by placing a back-slash before it to prevent it from being interpreted as the end of the format string. The format string may also contain the special control characters "\n" for new-line and "\t" for tab.

The `CustomLog` directive sets up a new log file using the defined *nickname*. The filename for the access log is relative to the [ServerRoot](#) unless it begins with a slash.

The above configuration will write log entries in a format known as the Common Log Format (CLF). This standard format can be produced by many different web servers and read by many log analysis programs. The log file entries produced in CLF will look something like this:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0"
200 2326
```

Each part of this log entry is described below.

127.0.0.1 (%h)

This is the IP address of the client (remote host) which made the request to the server. If [HostNameLookups](#) is set to On, then the server will try to determine the hostname and log it in place of the IP address. However, this configuration is not recommended since it can significantly slow the server. Instead, it is best to use a log post-processor such as [logresolve](#) to determine the hostnames. The IP address reported here is not necessarily the address of the machine at which the user is sitting. If a proxy server exists between the user and the server, this address will be the address of the proxy, rather than the originating machine.

- (%l)

The "hyphen" in the output indicates that the requested piece of information is not available. In this case, the information that is not available is the RFC 1413 identity of the client determined by `identd` on the clients machine. This information is highly unreliable and should almost never be used except on tightly controlled internal networks. Apache `httpd` will not even attempt to determine this information unless [IdentityCheck](#) is set to On.

frank (%u)

This is the userid of the person requesting the document as determined by HTTP authentication. The same value is typically provided to CGI scripts in the `REMOTE_USER` environment variable. If the status code for the request (see below) is 401, then this value should not be trusted because the user is not yet authenticated. If the document is not password protected, this entry will be "-" just like the previous one.

[10/Oct/2000:13:55:36 -0700] (%t)

The time that the server finished processing the request. The format is:

```
[day/month/year:hour:minute:second zone]
day = 2*digit
month = 3*letter
year = 4*digit
hour = 2*digit
minute = 2*digit
second = 2*digit
zone = ('+' | '-' ) 4*digit
```

It is possible to have the time displayed in another format by specifying `%{format}t` in the log format string, where `format` is as in `strftime(3)` from the C standard library.

"GET /apache_pb.gif HTTP/1.0" (\ "%r\")

The request line from the client is given in double quotes. The request line contains a great deal of useful information. First, the method used by the client is `GET`. Second, the client requested the resource `/apache_pb.gif`, and third, the client used the protocol `HTTP/1.0`. It is also possible to log one or more parts of the request line independently. For example, the

format string "%m %U%q %H" will log the method, path, query-string, and protocol, resulting in exactly the same output as "%r".

200 (%>s)

This is the status code that the server sends back to the client. This information is very valuable, because it reveals whether the request resulted in a successful response (codes beginning in 2), a redirection (codes beginning in 3), an error caused by the client (codes beginning in 4), or an error in the server (codes beginning in 5). The full list of possible status codes can be found in the [HTTP specification](#) (RFC2616 section 10).

2326 (%b)

The last entry indicates the size of the object returned to the client, not including the response headers. If no content was returned to the client, this value will be "-". To log "0" for no content, use %B instead.

Combined Log Format

Another commonly used format string is called the Combined Log Format. It can be used as follows.

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\" "
combined
CustomLog log/acces_log combined
```

This format is exactly the same as the Common Log Format, with the addition of two more fields. Each of the additional fields uses the percent-directive %*{header}*i, where *header* can be any HTTP request header. The access log under this format will look like:

```
127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700] "GET /apache_pb.gif HTTP/1.0"
200 2326 "http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I
;Nav) "
```

The additional fields are:

"http://www.example.com/start.html" ("%{Referer}i")

The "Referer" (sic) HTTP request header. This gives the site that the client reports having been referred from. (This should be the page that links to or includes /apache_pb.gif).

"Mozilla/4.08 [en] (Win98; I ;Nav)" ("%{User-agent}i")

The User-Agent HTTP request header. This is the identifying information that the client browser reports about itself.

Multiple Access Logs

Multiple access logs can be created simply by specifying multiple CustomLog directives in the configuration file. For example, the following directives will create three access logs. The first contains the basic CLF information, while the second and third contain referer and browser information. The last two CustomLog lines show how to mimic the effects of the ReferLog and AgentLog directives.

```
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common
CustomLog logs/referer_log "%{Referer}i -> %U"
CustomLog logs/agent_log "%{User-agent}i"
```

This example also shows that it is not necessary to define a nickname with the LogFormat directive. Instead, the log format can be specified directly in the CustomLog directive.

Conditional Logging

There are times when it is convenient to exclude certain entries from the access logs based on characteristics of the client request. This is easily accomplished with the help of [environment variables](#). First, an environment variable must be set to indicate that the request meets certain conditions. This is usually accomplished with [SetEnvIf](#). Then the env= clause of the CustomLog directive is used to include or exclude requests where the environment variable is set. Some examples:

```
# Mark requests from the loop-back interface
SetEnvIf Remote_Addr "127\.0\.0\.1" dontlog
# Mark requests for the robots.txt file
```

```
SetEnvIf Request_URI "^/robots\.txt$" dontlog
# Log what remains
CustomLog logs/access_log common env=!dontlog
```

As another example, consider logging requests from english-speakers to one log file, and non-english speakers to a different log file.

```
SetEnvIf Accept-Language "en" english
CustomLog logs/english_log common env=english
CustomLog logs/non_english_log common env=!english
```

Although we have just shown that conditional logging is very powerful and flexibly, it is not the only way to control the contents of the logs. Log files are more useful when they contain a complete record of server activity. It is often easier to simply post-process the log files to remove requests that you do not want to consider.

Log Rotation

On even a moderately busy server, the quantity of information stored in the log files is very large. The access log file typically grows 1 MB or more per 10,000 requests. It will consequently be necessary to periodically rotate the log files by moving or deleting the existing logs. This cannot be done while the server is running, because Apache will continue writing to the old log file as long as it holds the file open. Instead, the server must be [restarted](#) after the log files are moved or deleted so that it will open new log files.

By using a *graceful* restart, the server can be instructed to open new log files without losing any existing or pending connections from clients. However, in order to accomplish this, the server must continue to write to the old log files while it finishes serving old requests. It is therefore necessary to wait for some time after the restart before doing any processing on the log files. A typical scenario that simply rotates the logs and compresses the old logs to save space is:

```
mv access_log access_log.old
mv error_log error_log.old
apachectl graceful
sleep 600
gzip access_log.old error_log.old
```

Another way to perform log rotation is using [piped logs](#) as discussed in the next section.

Piped Logs

Apache httpd is capable of writing error and access log files through a pipe to another process, rather than directly to a file. This capability dramatically increases the flexibility of logging, without adding code to the main server. In order to write logs to a pipe, simply replace the filename with the pipe character "|", followed by the name of the executable which should accept log entries on its standard input. Apache will start the piped-log process when the server starts, and will restart it if it crashes while the server is running. (This last feature is why we can refer to this technique as "reliable piped logging".)

Piped log processes are spawned by the parent Apache httpd process, and inherit the userid of that process. This means that piped log programs usually run as root. It is therefore very important to keep the programs simple and secure.

Some simple examples using piped logs:

```
# compressed logs
CustomLog "|/usr/bin/gzip -c >> /var/log/access_log.gz" common
# almost-real-time name resolution
CustomLog "|/usr/local/apache/bin/logresolve >> /var/log/access_log" common
```

Notice that quotes are used to enclose the entire command that will be called for the pipe. Although these examples are for the access log, the same technique can be used for the error log.

One important use of piped logs is to allow log rotation without having to restart the server. The Apache HTTP Server includes a simple program called [rotatelog](#)s for this purpose. For example, to rotate the logs every 24 hours, you can use:

```
CustomLog "|/usr/local/apache/bin/rotatelog /var/log/access_log 86400"
```

A similar, but much more flexible log rotation program called [cronolog](#) is available at an external site.

As with conditional logging, piped logs are a very powerful tool, but they should not be used where a simpler solution like off-line post-processing is available.

Virtual Hosts

When running a server with many [virtual hosts](#), there are several options for dealing with log files. First, it is possible to use logs exactly as in a single-host server. Simply by placing the logging directives outside the <VirtualHost> sections in the main server context, it is possible to log all requests in the same access log and error log. This technique does not allow for easy collection of statistics on individual virtual hosts.

If CustomLog or ErrorLog directives are placed inside a <VirtualHost> section, all requests or errors for that virtual host will be logged only to the specified file. Any virtual host which does not have logging directives will still have its requests sent to the main server logs. This technique is very useful for a small number of virtual hosts, but if the number of hosts is very large, it can be complicated to manage. In addition, it can often create problems with [insufficient file descriptors](#).

For the access log, there is a very good compromise. By adding information on the virtual host to the log format string, it is possible to log all hosts to the same log, and later split the log into individual files. For example, consider the following directives.

```
LogFormat "%v %l %u %t \"%r\" %>s %b" comonvhost
CustomLog logs/access_log comonvhost
```

The %v is used to log the name of the virtual host that is serving the request. Then a program like [split-logfile](#) can be used to post-process the access log in order to split it into one file per virtual host.

Unfortunately, no similar technique is available for the error log, so you must choose between mixing all virtual hosts in the same error log and using one error log per virtual host.

Other Log Files

Related Modules	Related Directives
mod_cgi	PidFile
mod_rewrite	RewriteLog
	RewriteLogLevel
	ScriptLog
	ScriptLogLength
	ScriptLogBuffer

PID File

On startup, Apache httpd saves the process id of the parent httpd process to the file logs/httpd.pid. This filename can be changed with the [PidFile](#) directive. The process-id is for use by the administrator in restarting and terminating the daemon by sending signals to the parent process; on Windows, use the -k command line option instead. For more information see the [Stopping and Restarting](#) page.

Script Log

In order to aid in debugging, the [ScriptLog](#) directive allows you to record the input to and output from CGI scripts. This should only be used in testing - not for live servers. More information is available in the [mod_cgi documentation](#).

Rewrite Log

When using the powerful and complex features of [mod_rewrite](#), it is almost always necessary to use the [RewriteLog](#) to help in debugging. This log file produces a detailed analysis of how the rewriting engine transforms requests. The level of detail is controlled by the [RewriteLogLevel](#) directive.

Apache HTTP Server





Apache HTTP Server

Mapping URLs to Filesystem Locations

This document explains how Apache uses the URL of a request to determine the filesystem location from which to serve a file.

- [DocumentRoot](#)
- [Files Outside the DocumentRoot](#)
- [User Directories](#)
- [URL Redirection](#)
- [Rewrite Engine](#)
- [File Not Found](#)

Related Modules	Related Directives
mod_alias	Alias
mod_rewrite	AliasMatch
mod_userdir	CheckSpelling
mod_speling	DocumentRoot
mod_vhost_alias	ErrorDocument
	Options
	Redirect
	RedirectMatch
	RewriteCond
	RewriteRule
	ScriptAlias
	ScriptAliasMatch
	UserDir

DocumentRoot

In deciding what file to serve for a given request, Apache's default behavior is to take the URL-Path for the request (the part of the URL following the hostname and port) and add it to the end of the [DocumentRoot](#) specified in your configuration files. Therefore, the files and directories underneath the `DocumentRoot` make up the basic document tree which will be visible from the web.

Apache is also capable of [Virtual Hosting](#), where the server receives requests for more than one host. In this case, a different `DocumentRoot` can be specified for each virtual host, or alternatively, the directives provided by the module [mod_vhost_alias](#) can be used to dynamically determine the appropriate place from which to serve content based on the requested IP address or hostname.

Files Outside the DocumentRoot

There are frequently circumstances where it is necessary to allow web access to parts of the filesystem that are not strictly underneath the [DocumentRoot](#). Apache offers several different ways to accomplish this. On Unix systems, symbolic links can bring other parts of the filesystem under the `DocumentRoot`. For security reasons, Apache will follow symbolic links only if the

[Options](#) setting for the relevant directory includes `FollowSymLinks` or `SymLinksIfOwnerMatch`.

Alternatively, the [Alias](#) directive will map any part of the filesystem into the web space. For example, with

```
Alias /docs /var/web/
```

the URL `http://www.example.com/docs/dir/file.html` will be served from `/var/web/dir/file.html`. The [ScriptAlias](#) directive works the same way, with the additional effect that all content located at the target path is treated as CGI scripts.

For situations where you require additional flexibility, you can use the [AliasMatch](#) and [ScriptAliasMatch](#) directives to do powerful [regular-expression](#) based matching and substitution. For example,

```
ScriptAliasMatch ^/~([^/]+)/cgi-bin/(.*) /home/$1/cgi-bin/$2
```

will map a request to `http://example.com/~user/cgi-bin/script.cgi` to the path `/home/user/cgi-bin/script.cgi` and will treat the resulting file as a CGI script.

User Directories

Traditionally on Unix systems, the home directory of a particular *user* can be referred to as `~user/`. The module [mod_userdir](#) extends this idea to the web by allowing files under each user's home directory to be accessed using URLs such as the following.

```
http://www.example.com/~user/file.html
```

For security reasons, it is inappropriate to give direct access to a user's home directory from the web. Therefore, the [UserDir](#) directive specifies a directory underneath the user's home directory where web files are located. Using the default setting of `Userdir public_html`, the above URL maps to a file at a directory like `/home/user/public_html/file.html` where `/home/user/` is the user's home directory as specified in `/etc/passwd`.

There are also several other forms of the `Userdir` directive which you can use on systems where `/etc/passwd` does not contain the location of the home directory.

Some people find the "~" symbol (which is often encoded on the web as `%7e`) to be awkward and prefer to use an alternate string to represent user directories. This functionality is not supported by `mod_userdir`. However, if users' home directories are structured in a regular way, then it is possible to use the [AliasMatch](#) directive to achieve the desired effect. For example, to make `http://www.example.com/upages/user/file.html` map to `/home/user/public_html/file.html`, use the following `AliasMatch` directive:

```
AliasMatch ^/upages/([^/]+)/?(.*) /home/$1/public_html/$2
```

URL Redirection

The configuration directives discussed in the above sections tell Apache to get content from a specific place in the filesystem and return it to the client. Sometimes, it is desirable instead to inform the client that the requested content is located at a different URL, and instruct the client to make a new request with the new URL. This is called *redirection* and is implemented by the [Redirect](#) directive. For example, if the contents of the directory `/foo/` under the `DocumentRoot` are moved to the new directory `/bar/`, you can instruct clients to request the content at the new location as follows:

```
Redirect permanent /foo/ http://www.example.com/bar/
```

This will redirect any URL-Path starting in `/foo/` to the same URL path on the `www.example.com` server with `/bar/` substituted for `/foo/`. You can redirect clients to any server, not only the origin server.

Apache also provides a [RedirectMatch](#) directive for more complicated rewriting problems. For example, to redirect requests for the site home page to a different site, but leave all other requests alone, use the following configuration:

```
RedirectMatch permanent ^/$ http://www.example.com/startpage.html
```

Alternatively, to temporarily redirect all pages on a site to one particular page, use the following:

```
RedirectMatch temp .* http://www.example.com/startpage.html
```

Rewriting Engine

When even more powerful substitution is required, the rewriting engine provided by [mod_rewrite](#) can be useful. The directives provided by this module use characteristics of the request such as browser type or source IP address in deciding from where to serve content. In addition, `mod_rewrite` can use external database files or programs to determine how to handle a request. Many practical examples employing `mod_rewrite` are discussed in the [URL Rewriting Guide](#).

File Not Found

Inevitably, URLs will be requested for which no matching file can be found in the filesystem. This can happen for several reasons. In some cases, it can be a result of moving documents from one location to another. In this case, it is best to use [URL redirection](#) to inform clients of the new location of the resource. In this way, you can assure that old bookmarks and links will continue to work, even though the resource is at a new location.

Another common cause of "File Not Found" errors is accidental mistyping of URLs, either directly in the browser, or in HTML links. Apache provides the module [mod_speling](#) (sic) to help with this problem. When this module is activated, it will intercept "File Not Found" errors and look for a resource with a similar filename. If one such file is found, `mod_speling` will send an HTTP redirect to the client informing it of the correct location. If several "close" files are found, a list of available alternatives will be presented to the client.

An especially useful feature of `mod_speling`, is that it will compare filenames without respect to case. This can help systems where users are unaware of the case-sensitive nature of URLs and the unix filesystem. But using `mod_speling` for anything more than the occasional URL correction can place additional load on the server, since each "incorrect" request is followed by a URL redirection and a new request from the client.

If all attempts to locate the content fail, Apache returns an error page with HTTP status code 404 (file not found). The appearance of this page is controlled with the [ErrorDocument](#) directive and can be customized in a flexible manner as discussed in the [Custom error responses](#) and [International Server Error Responses](#) documents.

Apache HTTP Server





Apache HTTP Server Version 1.3

Security Tips for Server Configuration

- [Permissions on ServerRoot Directories](#)
- [Server Side Includes](#)
- [Non Script Aliased CGI](#)
- [Script Aliased CGI](#)
- [CGI in General](#)
- [Protecting System Settings](#)
- [Protect Server Files by Default](#)

Some hints and tips on security issues in setting up a web server. Some of the suggestions will be general, others specific to Apache.

Permissions on ServerRoot Directories

In typical operation, Apache is started by the root user, and it switches to the user defined by the [User](#) directive to serve hits. As is the case with any command that root executes, you must take care that it is protected from modification by non-root users. Not only must the files themselves be writeable only by root, but so must the directories, and parents of all directories. For example, if you choose to place ServerRoot in `/usr/local/apache` then it is suggested that you create that directory as root, with commands like these:

```
mkdir /usr/local/apache
cd /usr/local/apache
mkdir bin conf logs
chown 0 . bin conf logs
chgrp 0 . bin conf logs
chmod 755 . bin conf logs
```

It is assumed that `/`, `/usr`, and `/usr/local` are only modifiable by root. When you install the `httpd` executable, you should ensure that it is similarly protected:

```
cp httpd /usr/local/apache/bin
chown 0 /usr/local/apache/bin/httpd
chgrp 0 /usr/local/apache/bin/httpd
chmod 511 /usr/local/apache/bin/httpd
```

You can create an `htdocs` subdirectory which is modifiable by other users -- since root never executes any files out of there, and shouldn't be creating files in there.

If you allow non-root users to modify any files that root either executes or writes on then you open your system to root compromises. For example, someone could replace the `httpd` binary so that the next time you start it, it will execute some arbitrary code. If the `logs` directory is writeable (by a non-root user), someone could replace a log file with a symlink to some other system file, and then root might overwrite that file with arbitrary data. If the log files themselves are writeable (by a non-root user), then someone may be able to overwrite the log itself with bogus data.

Server Side Includes

Server Side Includes (SSI) present a server administrator with several potential security risks.

The first risk is the increased load on the server. All SSI-enabled files have to be parsed by Apache, whether or not there are any SSI directives included within the files. While this load increase is minor, in a shared server environment it can become significant.

SSI files also pose the same risks that are associated with CGI scripts in general. Using the "exec cmd" element, SSI-enabled files can execute any CGI script or program under the permissions of the user and group Apache runs as, as configured in httpd.conf. That should definitely give server administrators pause.

There are ways to enhance the security of SSI files while still taking advantage of the benefits they provide.

To isolate the damage a wayward SSI file can cause, a server administrator can enable [suexec](#) as described in the [CGI in General](#) section.

Enabling SSI for files with .html or .htm extensions can be dangerous. This is especially true in a shared, or high traffic, server environment. SSI-enabled files should have a separate extension, such as the conventional .shtml. This helps keep server load at a minimum and allows for easier management of risk.

Another solution is to disable the ability to run scripts and programs from SSI pages. To do this replace `Includes` with `IncludesNOEXEC` in the [Options](#) directive. Note that users may still use `<!--#include virtual="..." -->` to execute CGI scripts if these scripts are in directories designated by a [ScriptAlias](#) directive.

Non Script Aliased CGI

Allowing users to execute CGI scripts in any directory should only be considered if;

1. You trust your users not to write scripts which will deliberately or accidentally expose your system to an attack.
 2. You consider security at your site to be so feeble in other areas, as to make one more potential hole irrelevant.
 3. You have no users, and nobody ever visits your server.
-

Script Aliased CGI

Limiting CGI to special directories gives the admin control over what goes into those directories. This is inevitably more secure than non script aliased CGI, but **only if users with write access to the directories are trusted** or the admin is willing to test each new CGI script/program for potential security holes.

Most sites choose this option over the non script aliased CGI approach.

CGI in General

Always remember that you must trust the writers of the CGI script/programs or your ability to spot potential security holes in CGI, whether they were deliberate or accidental.

All the CGI scripts will run as the same user, so they have potential to conflict (accidentally or deliberately) with other scripts *e.g.* User A hates User B, so he writes a script to trash User B's CGI database. One program which can be used to allow scripts to run as different users is [suEXEC](#) which is included with Apache as of 1.2 and is called from special hooks in the Apache server code. Another popular way of doing this is with [CGIWrap](#).

Protecting System Settings

To run a really tight ship, you'll want to stop users from setting up `.htaccess` files which can override security features you've configured. Here's one way to do it...

In the server configuration file, put

```
<Directory />
AllowOverride None
Options None
Allow from all
</Directory>
```

Then setup for specific directories

This stops all overrides, Includes and accesses in all directories apart from those named.

Protect Server Files by Default

One aspect of Apache which is occasionally misunderstood is the feature of default access. That is, unless you take steps to change it, if the server can find its way to a file through normal URL mapping rules, it can serve it to clients.

For instance, consider the following example:

1. `# cd /; ln -s / public_html`
2. Accessing `http://localhost/~root/`

This would allow clients to walk through the entire filesystem. To work around this, add the following block to your server's configuration:

```
<Directory />
    Order Deny,Allow
    Deny from all
</Directory>
```

This will forbid default access to filesystem locations. Add appropriate [<Directory>](#) blocks to allow access only in those areas you wish. For example,

```
<Directory /usr/users/*/public_html>
    Order Deny,Allow
    Allow from all
</Directory>
<Directory /usr/local/httpd>
    Order Deny,Allow
    Allow from all
</Directory>
```

Pay particular attention to the interactions of [<Location>](#) and [<Directory>](#) directives; for instance, even if `<Directory />` denies access, a `<Location />` directive might overturn it.

Also be wary of playing games with the [UserDir](#) directive; setting it to something like `"/"` would have the same effect, for root, as the first example above. If you are using Apache 1.3 or above, we strongly recommend that you include the following line in your server configuration files:

```
UserDir disabled root
```

Please send any other useful security tips to The Apache Group by filling out a [problem report](#). If you are confident you have found a security bug in the Apache source code itself, [please let us know](#).

Apache HTTP Server Version 1.3





Apache HTTP Server

Apache 1.3 Dynamic Shared Object (DSO) Support

Originally written by
Ralf S. Engelschall <rse@apache.org>, April 1998

Background

On modern Unix derivatives there exists a nifty mechanism usually called dynamic linking/loading of *Dynamic Shared Objects* (DSO) which provides a way to build a piece of program code in a special format for loading it at run-time into the address space of an executable program.

This loading can usually be done in two ways: Automatically by a system program called `ld.so` when an executable program is started or manually from within the executing program via a programmatic system interface to the Unix loader through the system calls `dlopen()`/`dlsym()`.

In the first way the DSO's are usually called *shared libraries* or *DSO libraries* and named `libfoo.so` or `libfoo.so.1.2`. They reside in a system directory (usually `/usr/lib`) and the link to the executable program is established at build-time by specifying `-lfoo` to the linker command. This hard-codes library references into the executable program file so that at start-time the Unix loader is able to locate `libfoo.so` in `/usr/lib`, in paths hard-coded via linker-options like `-R` or in paths configured via the environment variable `LD_LIBRARY_PATH`. It then resolves any (yet unresolved) symbols in the executable program which are available in the DSO.

Symbols in the executable program are usually not referenced by the DSO (because it's a reusable library of general code) and hence no further resolving has to be done. The executable program has no need to do anything on its own to use the symbols from the DSO because the complete resolving is done by the Unix loader. (In fact, the code to invoke `ld.so` is part of the run-time startup code which is linked into every executable program which has been bound non-static). The advantage of dynamic loading of common library code is obvious: the library code needs to be stored only once, in a system library like `libc.so`, saving disk space for every program.

In the second way the DSO's are usually called *shared objects* or *DSO files* and can be named with an arbitrary extension (although the canonical name is `foo.so`). These files usually stay inside a program-specific directory and there is no automatically established link to the executable program where they are used. Instead the executable program manually loads the DSO at run-time into its address space via `dlopen()`. At this time no resolving of symbols from the DSO for the executable program is done. But instead the Unix loader automatically resolves any (yet unresolved) symbols in the DSO from the set of symbols exported by the executable program and its already loaded DSO libraries (especially all symbols from the ubiquitous `libc.so`). This way the DSO gets knowledge of the executable program's symbol set as if it had been statically linked with it in the first place.

Finally, to take advantage of the DSO's API the executable program has to resolve particular symbols from the DSO via `dlsym()` for later use inside dispatch tables *etc*. In other words: The executable program has to manually resolve every symbol it needs to be able to use it. The advantage of such a mechanism is that optional program parts need not be loaded (and thus do not spend memory) until they are needed by the program in question. When required, these program parts can be loaded dynamically to extend the base program's functionality.

Although this DSO mechanism sounds straightforward there is at least one difficult step here: The resolving of symbols from the executable program for the DSO when using a DSO to extend a program (the second way). Why? Because "reverse resolving" DSO symbols from the executable program's symbol set is against the library design (where the library has no knowledge about the programs it is used by) and is neither available under all platforms nor standardized. In practice the executable program's global symbols are often not re-exported and thus not available for

use in a DSO. Finding a way to force the linker to export all global symbols is the main problem one has to solve when using DSO for extending a program at run-time.

Windows and NetWare provide similar facilities, although they are implemented somewhat differently than the description of Unix DSO throughout this document. In particular, DSO modules (DLL's and NLM's, respectively) are built quite differently than their Unix cousins. This document does not attempt to explore the topic of building DSO modules on these platforms. The description of `mod_so` and its configuration, however, are similar.

Practical Usage

The shared library approach is the typical one, because it is what the DSO mechanism was designed for, hence it is used for nearly all types of libraries the operating system provides. On the other hand using shared objects for extending a program is not used by a lot of programs.

As of 1998 there are only a few software packages available which use the DSO mechanism to actually extend their functionality at run-time: Perl 5 (via its XS mechanism and the DynaLoader module), Netscape Server, *etc.* Starting with version 1.3, Apache joined the crew, because Apache already uses a module concept to extend its functionality and internally uses a dispatch-list-based approach to link external modules into the Apache core functionality. So, Apache is really predestined for using DSO to load its modules at run-time.

As of Apache 1.3, the configuration system supports two optional features for taking advantage of the modular DSO approach: compilation of the Apache core program into a DSO library for shared usage and compilation of the Apache modules into DSO files for explicit loading at run-time.

Implementation

The DSO support for loading individual Apache modules is based on a module named `mod_so.c` which has to be statically compiled into the Apache core. It is the only module besides `http_core.c` which cannot be put into a DSO itself (bootstrapping!). Practically all other distributed Apache modules then can then be placed into a DSO by individually enabling the DSO build for them via `configure's --enable-shared` option (see top-level `INSTALL` file) or by changing the `AddModule` command in your `src/Configuration` into a `SharedModule` command (see `src/INSTALL` file). After a module is compiled into a DSO named `mod_foo.so` you can use `mod_so's LoadModule` command in your `httpd.conf` file to load this module at server startup or restart.

To simplify this creation of DSO files for Apache modules (especially for third-party modules) a new support program named `apxs` (*APache eXtension*) is available. It can be used to build DSO based modules *outside of* the Apache source tree. The idea is simple: When installing Apache the `configure's make install` procedure installs the Apache C header files and puts the platform-dependent compiler and linker flags for building DSO files into the `apxs` program. This way the user can use `apxs` to compile his Apache module sources without the Apache distribution source tree and without having to fiddle with the platform-dependent compiler and linker flags for DSO support.

To place the complete Apache core program into a DSO library (only required on some of the supported platforms to force the linker to export the apache core symbols -- a prerequisite for the DSO modularization) the rule `SHARED_CORE` has to be enabled via `configure's --enable-rule=SHARED_CORE` option (see top-level `INSTALL` file) or by changing the `Rule` command in your `Configuration` file to `Rule SHARED_CORE=yes` (see `src/INSTALL` file). The Apache core code is then placed into a DSO library named `libhttpd.so`. Because one cannot link a DSO against static libraries on all platforms, an additional executable program named `libhttpd.ep` is created which both binds this static code and provides a stub for the `main()` function. Finally the `httpd` executable program itself is replaced by a bootstrapping code which automatically makes sure the Unix loader is able to load and start `libhttpd.ep` by providing the `LD_LIBRARY_PATH` to `libhttpd.so`.

Supported Platforms

Apache's `src/Configure` script currently has only limited but adequate built-in knowledge on how to compile DSO files, because as already mentioned this is heavily platform-dependent. Nevertheless all major Unix platforms are supported. The definitive current state (May 1999) is this:

- Out-of-the-box supported platforms:
(actually tested versions in parenthesis)
 - FreeBSD (2.1.5, 2.2.x, 3.x, 4.x)

- OpenBSD (2.x)
 - NetBSD (1.3.1)
 - BSDI (3.x, 4.x)
 - Linux (Debian/1.3.1, RedHat/4.2)
 - Solaris (2.4, 2.5, 2.6, 2.7)
 - SunOS (4.1.3)
 - Digital UNIX (4.0)
 - IRIX (5.3, 6.2)
 - HP/UX (10.20)
 - UnixWare (2.01, 2.1.2)
 - SCO (5.0.4)
 - AIX (3.2, 4.1.5, 4.2, 4.3)
 - ReliantUNIX/SINIX (5.43)
 - SVR4 (-)
 - Mac OS X Server (1.0)
 - Mac OS (10.0 preview 1)
 - OpenStep/Mach (4.2)
 - DGUX (??)
 - NetWare (5.1)
 - Windows (95, 98, NT 4.0, 2000)
- Explicitly unsupported platforms:
 - Ultrix (no dlopen-style interface under this platform)

Usage Summary

To give you an overview of the DSO features of Apache 1.3, here is a short and concise summary:

1. Placing the Apache core code (all the stuff which usually forms the `httpd` binary) into a DSO `libhttpd.so`, an executable program `libhttpd.ep` and a bootstrapping executable program `httpd` (Notice: this is only required on some of the supported platforms to force the linker to export the Apache core symbols, which in turn is a prerequisite for the DSO modularization):
 - Build and install via `configure` (preferred):

```
$ ./configure --prefix=/path/to/install
                --enable-rule=SHARED_CORE ...
$ make install
```

- Build and install manually:

```
- Edit src/Configuration:
<< Rule SHARED_CORE=default
>> Rule SHARED_CORE=yes
<< EXTRA_CFLAGS=
>> EXTRA_CFLAGS= -DSHARED_CORE_DIR="/path/to/install/libexec"
$ make
$ cp src/libhttpd.so* /path/to/install/libexec/
$ cp src/libhttpd.ep /path/to/install/libexec/
$ cp src/httpd /path/to/install/bin/
```

2. Build and install a *distributed* Apache module, say `mod_foo.c`, into its own DSO `mod_foo.so`:
 - Build and install via `configure` (preferred):

```
$ ./configure --prefix=/path/to/install
                --enable-shared=foo
$ make install
```

- Build and install manually:

```
- Edit src/Configuration:
<< AddModule      modules/xxxx/mod_foo.o
>> SharedModule modules/xxxx/mod_foo.so
$ make
$ cp src/xxxx/mod_foo.so /path/to/install/libexec
- Edit /path/to/install/etc/httpd.conf
>> LoadModule foo_module /path/to/install/libexec/mod_foo.so
```

3. Build and install a *third-party* Apache module, say `mod_foo.c`, into its own DSO `mod_foo.so`

- Build and install via `configure` (preferred):

```
$ ./configure --add-module=/path/to/3rdparty/mod_foo.c
--enable-shared=foo
$ make install
```

- Build and install manually:

```
$ cp /path/to/3rdparty/mod_foo.c /path/to/apache-1.3/src/modules/extra/
- Edit src/Configuration:
>> SharedModule modules/extra/mod_foo.so
$ make
$ cp src/xxxx/mod_foo.so /path/to/install/libexec
- Edit /path/to/install/etc/httpd.conf
>> LoadModule foo_module /path/to/install/libexec/mod_foo.so
```

4. Build and install a *third-party* Apache module, say `mod_foo.c`, into its own DSO `mod_foo.so` *outside of* the Apache source tree:

- Build and install via [apxs](#):

```
$ cd /path/to/3rdparty
$ apxs -c mod_foo.c
$ apxs -i -a -n foo mod_foo.so
```

Advantages & Disadvantages

The above DSO based features of Apache 1.3 have the following advantages:

- The server package is more flexible at run-time because the actual server process can be assembled at run-time via [LoadModule](#) `httpd.conf` configuration commands instead of `Configuration AddModule` commands at build-time. For instance this way one is able to run different server instances (standard & SSL version, minimalistic & powered up version [`mod_perl`, `PHP3`], *etc.*) with only one Apache installation.
- The server package can be easily extended with third-party modules even after installation. This is at least a great benefit for vendor package maintainers who can create a Apache core package and additional packages containing extensions like `PHP3`, `mod_perl`, `mod_fastcgi`, *etc.*
- Easier Apache module prototyping because with the DSO/apxs pair you can both work outside the Apache source tree and only need an `apxs -i` command followed by an `apachectl restart` to bring a new version of your currently developed module into the running Apache server.

DSO has the following disadvantages:

- The DSO mechanism cannot be used on every platform because not all operating systems support dynamic loading of code into the address space of a program.

- The server is approximately 20% slower at startup time because of the symbol resolving overhead the Unix loader now has to do.
- The server is approximately 5% slower at execution time under some platforms because position independent code (PIC) sometimes needs complicated assembler tricks for relative addressing which are not necessarily as fast as absolute addressing.
- Because DSO modules cannot be linked against other DSO-based libraries (`ld -lfoo`) on all platforms (for instance a.out-based platforms usually don't provide this functionality while ELF-based platforms do) you cannot use the DSO mechanism for all types of modules. Or in other words, modules compiled as DSO files are restricted to only use symbols from the Apache core, from the C library (`libc`) and all other dynamic or static libraries used by the Apache core, or from static library archives (`libfoo.a`) containing position independent code. The only chances to use other code is to either make sure the Apache core itself already contains a reference to it, loading the code yourself via `dlopen()` or enabling the `SHARED_CHAIN` rule while building Apache when your platform supports linking DSO files against DSO libraries.
- Under some platforms (many SVR4 systems) there is no way to force the linker to export all global symbols for use in DSO's when linking the Apache `httpd` executable program. But without the visibility of the Apache core symbols no standard Apache module could be used as a DSO. The only chance here is to use the `SHARED_CORE` feature because this way the global symbols are forced to be exported. As a consequence the Apache `src/Configure` script automatically enforces `SHARED_CORE` on these platforms when DSO features are used in the `Configuration` file or on the `configure` command line.

Apache HTTP Server





Apache HTTP Server

Content Negotiation

Apache's support for content negotiation has been updated to meet the HTTP/1.1 specification. It can choose the best representation of a resource based on the browser-supplied preferences for media type, languages, character set and encoding. It also implements a couple of features to give more intelligent handling of requests from browsers which send incomplete negotiation information.

Content negotiation is provided by the [mod_negotiation](#) module, which is compiled in by default.

About Content Negotiation

A resource may be available in several different representations. For example, it might be available in different languages or different media types, or a combination. One way of selecting the most appropriate choice is to give the user an index page, and let them select. However it is often possible for the server to choose automatically. This works because browsers can send as part of each request information about what representations they prefer. For example, a browser could indicate that it would like to see information in French, if possible, else English will do. Browsers indicate their preferences by headers in the request. To request only French representations, the browser would send

```
Accept-Language: fr
```

Note that this preference will only be applied when there is a choice of representations and they vary by language.

As an example of a more complex request, this browser has been configured to accept French and English, but prefer French, and to accept various media types, preferring HTML over plain text or other text types, and preferring GIF or JPEG over other media types, but also allowing any other media type as a last resort:

```
Accept-Language: fr; q=1.0, en; q=0.5
Accept: text/html; q=1.0, text/*; q=0.8, image/gif; q=0.6,
       image/jpeg; q=0.6, image/*; q=0.5, */*; q=0.1
```

Apache 1.2 supports 'server driven' content negotiation, as defined in the HTTP/1.1 specification. It fully supports the Accept, Accept-Language, Accept-Charset and Accept-Encoding request headers. Apache 1.3.4 also supports 'transparent' content negotiation, which is an experimental negotiation protocol defined in RFC 2295 and RFC 2296. It does not offer support for 'feature negotiation' as defined in these RFCs.

A **resource** is a conceptual entity identified by a URI (RFC 2396). An HTTP server like Apache provides access to **representations** of the resource(s) within its namespace, with each representation in the form of a sequence of bytes with a defined media type, character set, encoding, etc. Each resource may be associated with zero, one, or more than one representation at any given time. If multiple representations are available, the resource is referred to as **negotiable** and each of its representations is termed a **variant**. The ways in which the variants for a negotiable resource vary are called the **dimensions** of negotiation.

Negotiation in Apache

In order to negotiate a resource, the server needs to be given information about each of the variants. This is done in one of two ways:

- Using a type map (*i.e.*, a *.var file) which names the files containing the variants explicitly, or
- Using a 'MultiViews' search, where the server does an implicit filename pattern match and chooses from among the results.

Using a type-map file

A type map is a document which is associated with the handler named `type-map` (or, for backwards-compatibility with older Apache configurations, the mime type `application/x-type-map`). Note that to use this feature, you must have a handler set in the configuration that defines a file suffix as `type-map`; this is best done with a

```
AddHandler type-map .var
```

in the server configuration file. See the comments in the sample config file for more details.

Type map files have an entry for each available variant; these entries consist of contiguous HTTP-format header lines. Entries for different variants are separated by blank lines. Blank lines are illegal within an entry. It is conventional to begin a map file with an entry for the combined entity as a whole (although this is not required, and if present will be ignored). An example map file is:

```
URI: foo
```

```
URI: foo.en.html
Content-type: text/html
Content-language: en
```

```
URI: foo.fr.de.html
Content-type: text/html;charset=iso-8859-2
Content-language: fr, de
```

If the variants have different source qualities, that may be indicated by the "qs" parameter to the media type, as in this picture (available as jpeg, gif, or ASCII-art):

```
URI: foo
```

```
URI: foo.jpeg
Content-type: image/jpeg; qs=0.8
```

```
URI: foo.gif
Content-type: image/gif; qs=0.5
```

```
URI: foo.txt
Content-type: text/plain; qs=0.01
```

qs values can vary in the range 0.000 to 1.000. Note that any variant with a qs value of 0.000 will never be chosen. Variants with no 'qs' parameter value are given a qs factor of 1.0. The qs parameter indicates the relative 'quality' of this variant compared to the other available variants, independent of the client's capabilities. For example, a jpeg file is usually of higher source quality than an ascii file if it is attempting to represent a photograph. However, if the resource being represented is an original ascii art, then an ascii representation would have a higher source quality than a jpeg representation. A qs value is therefore specific to a given variant depending on the nature of the resource it represents.

The full list of headers recognized is:

URI :

uri of the file containing the variant (of the given media type, encoded with the given content encoding). These are interpreted as URLs relative to the map file; they must be on the same server (!), and they must refer to files to which the client would be granted access if they were to be requested directly.

Content-Type :

media type --- charset, level and "qs" parameters may be given. These are often referred to as MIME types; typical media types are `image/gif`, `text/plain`, or `text/html`; `level=3`.

Content-Language :

The languages of the variant, specified as an Internet standard language tag from RFC 1766 (*e.g.*, `en` for English, `kr` for Korean, *etc.*).

Content-Encoding :

If the file is compressed, or otherwise encoded, rather than containing the actual raw data, this says how that was done.

Apache only recognizes encodings that are defined by an [AddEncoding](#) directive. This normally includes the encodings `x-compress` for compress'd files, and `x-gzip` for gzip'd files. The `x-` prefix is ignored for encoding comparisons.

Content-Length:

The size of the file. Specifying content lengths in the type-map allows the server to compare file sizes without checking the actual files.

Description:

A human-readable textual description of the variant. If Apache cannot find any appropriate variant to return, it will return an error response which lists all available variants instead. Such a variant list will include the human-readable variant descriptions.

Multiviews

`MultiViews` is a per-directory option, meaning it can be set with an `Options` directive within a `<Directory>`, `<Location>` or `<Files>` section in `access.conf`, or (if `AllowOverride` is properly set) in `.htaccess` files. Note that `Options All` does not set `MultiViews`; you have to ask for it by name.

The effect of `MultiViews` is as follows: if the server receives a request for `/some/dir/foo`, if `/some/dir` has `MultiViews` enabled, and `/some/dir/foo` does *not* exist, then the server reads the directory looking for files named `foo.*`, and effectively fakes up a type map which names all those files, assigning them the same media types and content-encodings it would have if the client had asked for one of them by name. It then chooses the best match to the client's requirements.

`MultiViews` may also apply to searches for the file named by the `DirectoryIndex` directive, if the server is trying to index a directory. If the configuration files specify

```
DirectoryIndex index
```

then the server will arbitrate between `index.html` and `index.html3` if both are present. If neither are present, and `index.cgi` is there, the server will run it.

If one of the files found when reading the directive is a CGI script, it's not obvious what should happen. The code gives that case special treatment --- if the request was a POST, or a GET with `QUERY_ARGS` or `PATH_INFO`, the script is given an extremely high quality rating, and generally invoked; otherwise it is given an extremely low quality rating, which generally causes one of the other views (if any) to be retrieved.

The Negotiation Methods

After Apache has obtained a list of the variants for a given resource, either from a type-map file or from the filenames in the directory, it invokes one of two methods to decide on the 'best' variant to return, if any. It is not necessary to know any of the details of how negotiation actually takes place in order to use Apache's content negotiation features. However the rest of this document explains the methods used for those interested.

There are two negotiation methods:

1. **Server driven negotiation with the Apache algorithm** is used in the normal case. The Apache algorithm is explained in more detail below. When this algorithm is used, Apache can sometimes 'fiddle' the quality factor of a particular dimension to achieve a better result. The ways Apache can fiddle quality factors is explained in more detail below.
2. **Transparent content negotiation** is used when the browser specifically requests this through the mechanism defined in RFC 2295. This negotiation method gives the browser full control over deciding on the 'best' variant, the result is therefore dependent on the specific algorithms used by the browser. As part of the transparent negotiation process, the browser can ask Apache to run the 'remote variant selection algorithm' defined in RFC 2296.

Dimensions of Negotiation

Dimension

Notes

Media Type Browser indicates preferences with the `Accept` header field. Each item can have an associated quality factor. Variant description can also have a quality factor (the "qs" parameter).

Language	Browser indicates preferences with the Accept-Language header field. Each item can have a quality factor. Variants can be associated with none, one or more than one language.
Encoding	Browser indicates preference with the Accept-Encoding header field. Each item can have a quality factor.
Charset	Browser indicates preference with the Accept-Charset header field. Each item can have a quality factor. Variants can indicate a charset as a parameter of the media type.

Apache Negotiation Algorithm

Apache can use the following algorithm to select the 'best' variant (if any) to return to the browser. This algorithm is not further configurable. It operates as follows:

1. First, for each dimension of the negotiation, check the appropriate *Accept** header field and assign a quality to each variant. If the *Accept** header for any dimension implies that this variant is not acceptable, eliminate it. If no variants remain, go to step 4.
2. Select the 'best' variant by a process of elimination. Each of the following tests is applied in order. Any variants not selected at each test are eliminated. After each test, if only one variant remains, select it as the best match and proceed to step 3. If more than one variant remains, move on to the next test.
 1. Multiply the quality factor from the Accept header with the quality-of-source factor for this variant's media type, and select the variants with the highest value.
 2. Select the variants with the highest language quality factor.
 3. Select the variants with the best language match, using either the order of languages in the Accept-Language header (if present), or else the order of languages in the LanguagePriority directive (if present).
 4. Select the variants with the highest 'level' media parameter (used to give the version of text/html media types).
 5. Select variants with the best charset media parameters, as given on the Accept-Charset header line. Charset ISO-8859-1 is acceptable unless explicitly excluded. Variants with a `text/*` media type but not explicitly associated with a particular charset are assumed to be in ISO-8859-1.
 6. Select those variants which have associated charset media parameters that are *not* ISO-8859-1. If there are no such variants, select all variants instead.
 7. Select the variants with the best encoding. If there are variants with an encoding that is acceptable to the user-agent, select only these variants. Otherwise if there is a mix of encoded and non-encoded variants, select only the unencoded variants. If either all variants are encoded or all variants are not encoded, select all variants.
 8. Select the variants with the smallest content length.
 9. Select the first variant of those remaining. This will be either the first listed in the type-map file, or when variants are read from the directory, the one whose file name comes first when sorted using ASCII code order.
3. The algorithm has now selected one 'best' variant, so return it as the response. The HTTP response header Vary is set to indicate the dimensions of negotiation (browsers and caches can use this information when caching the resource). End.
4. To get here means no variant was selected (because none are acceptable to the browser). Return a 406 status (meaning "No acceptable representation") with a response body consisting of an HTML document listing the available variants. Also set the HTTP Vary header to indicate the dimensions of variance.

Fiddling with Quality Values

Apache sometimes changes the quality values from what would be expected by a strict interpretation of the Apache negotiation algorithm above. This is to get a better result from the algorithm for browsers which do not send full or accurate information. Some of the most popular browsers send Accept header information which would otherwise result in the selection of the wrong variant in many cases. If a browser sends full and correct information these fiddles will not be applied.

Media Types and Wildcards

The Accept: request header indicates preferences for media types. It can also include 'wildcard' media types, such as "image/*" or "*/*" where the * matches any string. So a request including:

```
Accept: image/*, */*
```

would indicate that any type starting "image/" is acceptable, as is any other type (so the first "image/*" is redundant). Some browsers routinely send wildcards in addition to explicit types they can handle. For example:

```
Accept: text/html, text/plain, image/gif, image/jpeg, */*
```

The intention of this is to indicate that the explicitly listed types are preferred, but if a different representation is available, that is ok too. However under the basic algorithm, as given above, the `*/*` wildcard has exactly equal preference to all the other types, so they are not being preferred. The browser should really have sent a request with a lower quality (preference) value for `*.*`, such as:

```
Accept: text/html, text/plain, image/gif, image/jpeg, */*; q=0.01
```

The explicit types have no quality factor, so they default to a preference of 1.0 (the highest). The wildcard `*/*` is given a low preference of 0.01, so other types will only be returned if no variant matches an explicitly listed type.

If the `Accept:` header contains *no* `q` factors at all, Apache sets the `q` value of `*/*`, if present, to 0.01 to emulate the desired behavior. It also sets the `q` value of wildcards of the format `"type/*"` to 0.02 (so these are preferred over matches against `*/*`). If any media type on the `Accept:` header contains a `q` factor, these special values are *not* applied, so requests from browsers which send the correct information to start with work as expected.

Variants with no Language

If some of the variants for a particular resource have a language attribute, and some do not, those variants with no language are given a very low language quality factor of 0.001.

The reason for setting this language quality factor for variant with no language to a very low value is to allow for a default variant which can be supplied if none of the other variants match the browser's language preferences. For example, consider the situation with three variants:

- `foo.en.html`, language en
- `foo.fr.html`, language en
- `foo.html`, no language

The meaning of a variant with no language is that it is always acceptable to the browser. If the request `Accept-Language` header includes either en or fr (or both) one of `foo.en.html` or `foo.fr.html` will be returned. If the browser does not list either en or fr as acceptable, `foo.html` will be returned instead.

Extensions to Transparent Content Negotiation

Apache extends the transparent content negotiation protocol (RFC 2295) as follows. A new `{encoding . . }` element is used in variant lists to label variants which are available with a specific content-encoding only. The implementation of the RVSA/1.0 algorithm (RFC 2296) is extended to recognize encoded variants in the list, and to use them as candidate variants whenever their encodings are acceptable according to the `Accept-Encoding` request header. The RVSA/1.0 implementation does not round computed quality factors to 5 decimal places before choosing the best variant.

Note on hyperlinks and naming conventions

If you are using language negotiation you can choose between different naming conventions, because files can have more than one extension, and the order of the extensions is normally irrelevant (see [mod_mime](#) documentation for details).

A typical file has a MIME-type extension (*e.g.*, `html`), maybe an encoding extension (*e.g.*, `gz`), and of course a language extension (*e.g.*, `en`) when we have different language variants of this file.

Examples:

- `foo.en.html`
- `foo.html.en`
- `foo.en.html.gz`

Here some more examples of filenames together with valid and invalid hyperlinks:

Filename	Valid hyperlink	Invalid hyperlink
<i>foo.html.en</i>	foo foo.html	-
<i>foo.en.html</i>	foo	foo.html
<i>foo.html.en.gz</i>	foo foo.html	foo.gz foo.html.gz
<i>foo.en.html.gz</i>	foo	foo.html foo.html.gz foo.gz
<i>foo.gz.html.en</i>	foo foo.gz foo.gz.html	foo.html
<i>foo.html.gz.en</i>	foo foo.html foo.html.gz	foo.gz

Looking at the table above you will notice that it is always possible to use the name without any extensions in an hyperlink (*e.g.*, foo). The advantage is that you can hide the actual type of a document resp. file and can change it later, *e.g.*, from html to shtml or cgi without changing any hyperlink references.

If you want to continue to use a MIME-type in your hyperlinks (*e.g.* foo.html) the language extension (including an encoding extension if there is one) must be on the right hand side of the MIME-type extension (*e.g.*, foo.html.en).

Note on Caching

When a cache stores a representation, it associates it with the request URL. The next time that URL is requested, the cache can use the stored representation. But, if the resource is negotiable at the server, this might result in only the first requested variant being cached and subsequent cache hits might return the wrong response. To prevent this, Apache normally marks all responses that are returned after content negotiation as non-cacheable by HTTP/1.0 clients. Apache also supports the HTTP/1.1 protocol features to allow caching of negotiated responses.

For requests which come from a HTTP/1.0 compliant client (either a browser or a cache), the directive `CacheNegotiatedDocs` can be used to allow caching of responses which were subject to negotiation. This directive can be given in the server config or virtual host, and takes no arguments. It has no effect on requests from HTTP/1.1 clients.

Apache HTTP Server





Apache HTTP Server

Custom error responses

Purpose

Additional functionality. Allows webmasters to configure the response of Apache to some error or problem.

Customizable responses can be defined to be activated in the event of a server detected error or problem.

e.g. if a script crashes and produces a "500 Server Error" response, then this response can be replaced with either some friendlier text or by a redirection to another URL (local or external).

Old behavior

NCSA httpd 1.3 would return some boring old error/problem message which would often be meaningless to the user, and would provide no means of logging the symptoms which caused it.

New behavior

The server can be asked to;

1. Display some other text, instead of the NCSA hard coded messages, or
2. redirect to a local URL, or
3. redirect to an external URL.

Redirecting to another URL can be useful, but only if some information can be passed which can then be used to explain and/or log the error/problem more clearly.

To achieve this, Apache will define new CGI-like environment variables, *e.g.*

```
REDIRECT_HTTP_ACCEPT=*/, image/gif, image/x-xbitmap, image/jpeg
REDIRECT_HTTP_USER_AGENT=Mozilla/1.1b2 (X11; I; HP-UX A.09.05 9000/712)
REDIRECT_PATH=./bin:/usr/local/bin:/etc
REDIRECT_QUERY_STRING=
REDIRECT_REMOTE_ADDR=121.345.78.123
REDIRECT_REMOTE_HOST=ooh.ahhh.com
REDIRECT_SERVER_NAME=crash.bang.edu
REDIRECT_SERVER_PORT=80
REDIRECT_SERVER_SOFTWARE=Apache/0.8.15
REDIRECT_URL=/cgi-bin/buggy.pl
```

note the REDIRECT_ prefix.

At least REDIRECT_URL and REDIRECT_QUERY_STRING will be passed to the new URL (assuming it's a cgi-script or a cgi-include). The other variables will exist only if they existed prior to the error/problem. **None** of these will be set if your ErrorDocument is an *external* redirect (*i.e.*, anything starting with a scheme name like `http:`, even if it refers to the same host as the server).

Configuration

Use of "ErrorDocument" is enabled for .htaccess files when the ["FileInfo" override](#) is allowed.

Here are some examples...

```
ErrorDocument 500 /cgi-bin/crash-recover
ErrorDocument 500 "Sorry, our script crashed. Oh dear
ErrorDocument 500 http://xxx/
ErrorDocument 404 /Lame_excuses/not_found.html
```

```
ErrorDocument 401 /Subscription/how_to_subscribe.html
```

The syntax is,

[ErrorDocument](#) <3-digit-code> action

where the action can be,

1. Text to be displayed. Prefix the text with a quote ("). Whatever follows the quote is displayed. *Note: the (") prefix isn't displayed.*
 2. An external URL to redirect to.
 3. A local URL to redirect to.
-

Custom error responses and redirects

Purpose

Apache's behavior to redirected URLs has been modified so that additional environment variables are available to a script/server-include.

Old behavior

Standard CGI vars were made available to a script which has been redirected to. No indication of where the redirection came from was provided.

New behavior

A new batch of environment variables will be initialized for use by a script which has been redirected to. Each new variable will have the prefix REDIRECT_. REDIRECT_ environment variables are created from the CGI environment variables which existed prior to the redirect, they are renamed with a REDIRECT_ prefix, *i.e.*, HTTP_USER_AGENT becomes REDIRECT_HTTP_USER_AGENT. In addition to these new variables, Apache will define REDIRECT_URL and REDIRECT_STATUS to help the script trace its origin. Both the original URL and the URL being redirected to can be logged in the access log.

If the ErrorDocument specifies a local redirect to a CGI script, the script should include a "Status:" header field in its output in order to ensure the propagation all the way back to the client of the error condition that caused it to be invoked. For instance, a Perl ErrorDocument script might include the following:

```
:  
print "Content-type: text/html\n";  
printf "Status: %s Condition Intercepted\n", $ENV{"REDIRECT_STATUS"};  
:
```

If the script is dedicated to handling a particular error condition, such as 404 Not Found, it can use the specific code and error text instead.

Apache HTTP Server





Apache HTTP Server

Setting which addresses and ports Apache uses

When Apache starts, it connects to some port and address on the local machine and waits for incoming requests. By default, it listens to all addresses on the machine, and to the port as specified by the `Port` directive in the server configuration. However, it can be told to listen to more than one port, or to listen to only selected addresses, or a combination. This is often combined with the Virtual Host feature which determines how Apache responds to different IP addresses, hostnames and ports.

There are two directives used to restrict or specify which addresses and ports Apache listens to.

- [BindAddress](#) is used to restrict the server to listening to a single address, and can be used to permit multiple Apache servers on the same machine listening to different IP addresses.
- [Listen](#) can be used to make a single Apache server listen to more than one address and/or port.

BindAddress

Syntax: `BindAddress [* / IP-address / hostname]`

Default: `BindAddress *`

Context: server config

Status: Core

Makes the server bind to just the specified address. If the argument is `*` (an asterisk), the server binds to all interfaces currently marked as up on the server. The port bound to is set with the `Port` directive. Only one `BindAddress` should be used.

Listen

Syntax: `Listen [port / IP-address:port]`

Default: none

Context: server config

Status: Core

`Listen` can be used instead of `BindAddress` and `Port`. It tells the server to accept incoming requests (to listen) on the specified port or address-and-port combination. If the first format is used, with a port number only, the server listens on the given port on all interfaces marked as up, instead of the port given by the `Port` directive. If an IP address is given as well as a port, the server will listen on the given port and interface.

Multiple `Listen` directives may be used to specify a number of addresses and ports to listen to. The server will respond to requests from any of the listed addresses and ports.

For example, to make the server accept connections on both port 80 and port 8000, use:

```
Listen 80
Listen 8000
```

To make the server accept connections on two specified interfaces and port numbers, use

```
Listen 192.170.2.1:80
Listen 192.170.2.5:8000
```

How this works with Virtual Hosts

`BindAddress` and `Listen` do not implement Virtual Hosts. They tell the main Apache daemon process what addresses and ports to bind and listen on. If no `<VirtualHost>` directives are used, the server will behave the same for all accepted requests. However, `<VirtualHost>` can be used to specify a different behavior for one or more of the addresses and ports. To implement a `VirtualHost`, the server must:

- Be told to `Listen` to the desired address and port
- Have a `<VirtualHost>` section created for the specified address and port to set the behavior of this virtual host

Note that if the `<VirtualHost>` is set for an address and port that the server is not listening to, it cannot be accessed.

See also

See also the documentation on [Virtual Hosts](#), [BindAddress directive](#), [Port directive](#), [DNS Issues](#) and [<VirtualHost> section](#).

Apache HTTP Server





Apache HTTP Server

Environment Variables in Apache

The Apache HTTP Server provides a mechanism for storing information in named variables that are called *environment variables*. This information can be used to control various operations such as logging or access control. The variables are also used as a mechanism to communicate with external programs such as CGI scripts. This document discusses different ways to manipulate and use these variables.

Although these variables are referred to as *environment variables*, they are not the same as the environment variables controlled by the underlying operating system. Instead, these variables are stored and manipulated in an internal Apache structure. They only become actual operating system environment variables when they are provided to CGI scripts and Server Side Include scripts. If you wish to manipulate the operating system environment under which the server itself runs, you must use the standard environment manipulation mechanisms provided by your operating system shell.

- [Setting Environment Variables](#)
- [Using Environment Variables](#)
- [Special Purpose Environment Variables](#)
- [Examples](#)

Setting Environment Variables

Related Modules	Related Directives
mod_env	BrowserMatch
mod_rewrite	BrowserMatchNoCase
mod_setenvif	PassEnv
mod_unique_id	RewriteRule
	SetEnv
	SetEnvIf
	SetEnvIfNoCase
	UnsetEnv

Basic Environment Manipulation

The most basic way to set an environment variable in Apache is using the unconditional `SetEnv` directive. Variables may also be passed from the environment of the shell which started the server using the `PassEnv` directive.

Conditional Per-Request Settings

For additional flexibility, the directives provided by `mod_setenvif` allow environment variables to be set on a per-request basis, conditional on characteristics of particular requests. For example, a variable could be set only when a specific browser (User-Agent) is making a request, or only when a specific Referer [sic] header is found. Even more flexibility is available through the `mod_rewrite`'s `RewriteRule` which uses the `[E=...]` option to set environment variables.

Unique Identifiers

Finally, `mod_unique_id` sets the environment variable `UNIQUE_ID` for each request to a value which is guaranteed to be unique across "all" requests under very specific conditions.

Standard CGI Variables

In addition to all environment variables set within the Apache configuration and passed from the shell, CGI scripts and SSI pages are provided with a set of environment variables containing meta-information about the request as required by the [CGI specification](#).

Some Caveats

- It is not possible to override or change the standard CGI variables using the environment manipulation directives.
- When [suexec](#) is used to launch CGI scripts, the environment will be cleaned down to a set of *safe* variables before CGI scripts are launched. The list of *safe* variables is defined at compile-time in `suexec.c`.
- For portability reasons, the names of environment variables may contain only letters, numbers, and the underscore character. In addition, the first character may not be a number. Characters which do not match this restriction will be replaced by an underscore when passed to CGI scripts and SSI pages.

Using Environment Variables

Related Modules	Related Directives
mod_access	Allow
mod_cgi	CustomLog
mod_include	Deny
mod_log_config	LogFormat
mod_rewrite	RewriteCond
	RewriteRule

CGI Scripts

One of the primary uses of environment variables is to communicate information to CGI scripts. As discussed above, the environment passed to CGI scripts includes standard meta-information about the request in addition to any variables set within the Apache configuration. For more details, see the [CGI tutorial](#).

SSI Pages

Server-parsed (SSI) documents processed by `mod_include`'s `server-parsed` handler can print environment variables using the `echo` element, and can use environment variables in flow control elements to make parts of a page conditional on characteristics of a request. Apache also provides SSI pages with the standard CGI environment variables as discussed above. For more details, see the [SSI tutorial](#).

Access Control

Access to the server can be controlled based on the value of environment variables using the `allow from env=` and `deny from env=` directives. In combination with `SetEnvIf`, this allows for flexible control of access to the server based on characteristics of the client. For example, you can use these directives to deny access to a particular browser (User-Agent).

Conditional Logging

Environment variables can be logged in the access log using the `LogFormat` option `%e`. In addition, the decision on whether or not to log requests can be made based on the status of environment variables using the conditional form of the `CustomLog` directive. In combination with `SetEnvIf` this allows for flexible control of which requests are logged. For example, you can choose not to log requests for filenames ending in `gif`, or you can choose to only log requests from clients which are outside your subnet.

URL Rewriting

The `%{ENV:...}` form of *TestString* in the `RewriteCond` allows `mod_rewrite`'s rewrite engine to make decisions conditional on environment variables. Note that the variables accessible in `mod_rewrite` without the `ENV:` prefix are not actually environment variables. Rather, they are variables special to `mod_rewrite` which cannot be accessed from other modules.

Special Purpose Environment Variables

Interoperability problems have led to the introduction of mechanisms to modify the way Apache behaves when talking to particular clients. To make these mechanisms as flexible as possible, they are invoked by defining environment variables, typically with [BrowserMatch](#), though [SetEnv](#) and [PassEnv](#) could also be used, for example.

downgrade-1.0

This forces the request to be treated as a HTTP/1.0 request even if it was in a later dialect.

force-no-vary

This causes any `Vary` fields to be removed from the response header before it is sent back to the client. Some clients don't interpret this field correctly (see the [known client problems](#) page); setting this variable can work around this problem. Setting this variable also implies **force-response-1.0**.

force-response-1.0

This forces an HTTP/1.0 response when set. It was originally implemented as a result of a problem with AOL's proxies. Some clients may not behave correctly when given an HTTP/1.1 response, and this can be used to interoperate with them.

nokeepalive

This disables [KeepAlive](#) when set.

Examples

Changing protocol behavior with misbehaving clients

We recommend that the following lines be included in `httpd.conf` to deal with known client problems.

```
#
# The following directives modify normal HTTP response behavior.
# The first directive disables keepalive for Netscape 2.x and browsers that
# spoof it. There are known problems with these browser implementations.
# The second directive is for Microsoft Internet Explorer 4.0b2
```

```
# which has a broken HTTP/1.1 implementation and does not properly
# support keepalive when it is used on 301 or 302 (redirect) responses.
#
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 force-response-1.0

#
# The following directive disables HTTP/1.1 responses to browsers which
# are in violation of the HTTP/1.0 spec by not being able to grok a
# basic 1.1 response.
#
BrowserMatch "RealPlayer 4\.0" force-response-1.0
BrowserMatch "Java/1\.0" force-response-1.0
BrowserMatch "JDK/1\.0" force-response-1.0
```

Do not log requests for images in the access log

This example keeps requests for images from appearing in the access log. It can be easily modified to prevent logging of particular directories, or to prevent logging of requests coming from particular hosts.

```
SetEnvIf Request_URI \.gif image-request
SetEnvIf Request_URI \.jpg image-request
SetEnvIf Request_URI \.png image-request
CustomLog logs/access_log env=!image-request
```

Prevent "Image Theft"

This example shows how to keep people not on your server from using images on your server as inline-images on their pages. This is not a recommended configuration, but it can work in limited circumstances. We assume that all your images are in a directory called /web/images.

```
SetEnvIf Referer "^http://www.example.com/" local_referal
# Allow browsers that do not send Referer info
SetEnvIf Referer "^$" local_referal
<Directory /web/images>
    Order Deny,Allow
    Deny from all
    Allow from env=local_referal
</Directory>
```

Note: spelling of 'referer' and 'referral' is intentional.

For more information about this technique, see the ApacheToday tutorial "[Keeping Your Images from Adorning Other Sites](#)".

Apache HTTP Server





Apache HTTP Server

Apache's Handler Use

- [What is a Handler](#)
- [Examples](#)
- [Programmer's Note](#)

What is a Handler

Related Modules	Related Directives
mod_actions	Action
mod_asis	AddHandler
mod_cgi	RemoveHandler
mod_imap	SetHandler
mod_info	
mod_include	
mod_mime	
mod_negotiation	
mod_status	

A "handler" is an internal Apache representation of the action to be performed when a file is called. Generally, files have implicit handlers, based on the file type. Normally, all files are simply served by the server, but certain file types are "handled" separately.

Apache 1.1 adds the ability to use handlers explicitly. Based on either filename extensions or on location, handlers can be specified without relation to file type. This is advantageous both because it is a more elegant solution, and because it also allows for both a type **and** a handler to be associated with a file. (See also [Files with Multiple Extensions](#).)

Handlers can either be built into the server or included in a module, or they can be added with the [Action](#) directive. The built-in handlers in the standard distribution are as follows:

- **default-handler**: Send the file using the `default_handler()`, which is the handler used by default to handle static content. (core)
- **send-as-is**: Send file with HTTP headers as is. ([mod_asis](#))
- **cgi-script**: Treat the file as a CGI script. ([mod_cgi](#))
- **imap-file**: Parse as an imagemap rule file. ([mod_imap](#))
- **server-info**: Get the server's configuration information. ([mod_info](#))
- **server-parsed**: Parse for server-side includes. ([mod_include](#))
- **server-status**: Get the server's status report. ([mod_status](#))
- **type-map**: Parse as a type map file for content negotiation. ([mod_negotiation](#))

Examples

Modifying static content using a CGI script

The following directives will cause requests for files with the `html` extension to trigger the launch of the `footer.pl` CGI script.

```
Action add-footer /cgi-bin/footer.pl
AddHandler add-footer .html
```

Then the CGI script is responsible for sending the originally requested document (pointed to by the `PATH_TRANSLATED` environment variable) and making whatever modifications or additions are desired.

Files with HTTP headers

The following directives will enable the `send-as-is` handler, which is used for files which contain their own HTTP headers. All files in the `/web/htdocs/asis/` directory will be processed by the `send-as-is` handler, regardless of their filename extensions.

```
<Directory /web/htdocs/asis>
SetHandler send-as-is
</Directory>
```

Programmer's Note

In order to implement the handler features, an addition has been made to the [Apache API](#) that you may wish to make use of. Specifically, a new record has been added to the `request_rec` structure:

```
char *handler
```

If you wish to have your module engage a handler, you need only to set `r->handler` to the name of the handler at any time prior to the `invoke_handler` stage of the request. Handlers are implemented as they were before, albeit using the handler name instead of a content type. While it is not necessary, the naming convention for handlers is to use a dash-separated word, with no slashes, so as to not invade the media type name-space.

Apache HTTP Server





Apache HTTP Server

Apache suEXEC Support

1. **CONTENTS**
2. [What is suEXEC?](#)
3. [Before we begin.](#)
4. [suEXEC Security Model.](#)
5. [Configuring & Installing suEXEC](#)
6. [Enabling & Disabling suEXEC](#)
7. [Using suEXEC](#)
8. [Debugging suEXEC](#)
9. [Beware the Jabberwock: Warnings & Examples](#)

What is suEXEC?

The **suEXEC** feature -- introduced in Apache 1.2 -- provides Apache users the ability to run **CGI** and **SSI** programs under user IDs different from the user ID of the calling web-server. Normally, when a CGI or SSI program executes, it runs as the same user who is running the web server.

Used properly, this feature can reduce considerably the security risks involved with allowing users to develop and run private CGI or SSI programs. However, if suEXEC is improperly configured, it can cause any number of problems and possibly create new holes in your computer's security. If you aren't familiar with managing **setuid** root programs and the security issues they present, we highly recommend that you not consider using suEXEC.

[BACK TO CONTENTS](#)

Before we begin.

Before jumping head-first into this document, you should be aware of the assumptions made on the part of the Apache Group and this document.

First, it is assumed that you are using a UNIX derivate operating system that is capable of **setuid** and **setgid** operations. All command examples are given in this regard. Other platforms, if they are capable of supporting suEXEC, may differ in their configuration.

Second, it is assumed you are familiar with some basic concepts of your computer's security and its administration. This involves an understanding of **setuid/setgid** operations and the various effects they may have on your system and its level of security.

Third, it is assumed that you are using an **unmodified** version of suEXEC code. All code for suEXEC has been carefully scrutinized and tested by the developers as well as numerous beta testers. Every precaution has been taken to ensure a simple yet solidly safe base of code. Altering this code can cause unexpected problems and new security risks. It is **highly** recommended you not alter the suEXEC code unless you are well versed in the particulars of security programming and are willing to share your work with the Apache Group for consideration.

Fourth, and last, it has been the decision of the Apache Group to **NOT** make suEXEC part of the default installation of Apache. To this end, suEXEC configuration requires of the administrator careful attention to details. After due consideration has been given to the various settings for suEXEC, the administrator may install suEXEC through normal installation methods. The values for these

settings need to be carefully determined and specified by the administrator to properly maintain system security during the use of suEXEC functionality. It is through this detailed process that the Apache Group hopes to limit suEXEC installation only to those who are careful and determined enough to use it.

Still with us? Yes? Good. Let's move on!

[BACK TO CONTENTS](#)

suEXEC Security Model

Before we begin configuring and installing suEXEC, we will first discuss the security model you are about to implement. By doing so, you may better understand what exactly is going on inside suEXEC and what precautions are taken to ensure your system's security.

suEXEC is based on a setuid "wrapper" program that is called by the main Apache web server. This wrapper is called when an HTTP request is made for a CGI or SSI program that the administrator has designated to run as a userid other than that of the main server. When such a request is made, Apache provides the suEXEC wrapper with the program's name and the user and group IDs under which the program is to execute.

The wrapper then employs the following process to determine success or failure -- if any one of these conditions fail, the program logs the failure and exits with an error, otherwise it will continue:

1. Was the wrapper called with the proper number of arguments?

The wrapper will only execute if it is given the proper number of arguments. The proper argument format is known to the Apache web server. If the wrapper is not receiving the proper number of arguments, it is either being hacked, or there is something wrong with the suEXEC portion of your Apache binary.

2. Is the user executing this wrapper a valid user of this system?

This is to ensure that the user executing the wrapper is truly a user of the system.

3. Is this valid user allowed to run the wrapper?

Is this user the user allowed to run this wrapper? Only one user (the Apache user) is allowed to execute this program.

4. Does the target program have an unsafe hierarchical reference?

Does the target program contain a leading '/' or have a '..' backreference? These are not allowed; the target program must reside within the Apache webspace.

5. Is the target user name valid?

Does the target user exist?

6. Is the target group name valid?

Does the target group exist?

7. Is the target user *NOT* superuser?

Presently, suEXEC does not allow 'root' to execute CGI/SSI programs.

8. Is the target userid *ABOVE* the minimum ID number?

The minimum user ID number is specified during configuration. This allows you to set the lowest possible userid that will be allowed to execute CGI/SSI programs. This is useful to block out "system" accounts.

9. Is the target group *NOT* the superuser group?

Presently, suEXEC does not allow the 'root' group to execute CGI/SSI programs.

10. Is the target groupid *ABOVE* the minimum ID number?

The minimum group ID number is specified during configuration. This allows you to set the lowest possible groupid that will be allowed to execute CGI/SSI programs. This is useful to block out "system" groups.

11. Can the wrapper successfully become the target user and group?

Here is where the program becomes the target user and group via setuid and setgid calls. The group access list is also initialized with all of the groups of which the user is a member.

12. Does the directory in which the program resides exist?

If it doesn't exist, it can't very well contain files.

13. Is the directory within the Apache webspace?

If the request is for a regular portion of the server, is the requested directory within the server's document root? If the request is for a UserDir, is the requested directory within the user's document root?

14. Is the directory *NOT* writable by anyone else?

We don't want to open up the directory to others; only the owner user may be able to alter this directories contents.

15. Does the target program exist?

If it doesn't exist, it can't very well be executed.

16. Is the target program *NOT* writable by anyone else?

We don't want to give anyone other than the owner the ability to change the program.

17. Is the target program *NOT* setuid or setgid?

We do not want to execute programs that will then change our UID/GID again.

18. Is the target user/group the same as the program's user/group?

Is the user the owner of the file?

19. Can we successfully clean the process environment to ensure safe operations?

suEXEC cleans the process' environment by establishing a safe execution PATH (defined during configuration), as well as only passing through those variables whose names are listed in the safe environment list (also created during configuration).

20. Can we successfully become the target program and execute?

Here is where suEXEC ends and the target program begins.

This is the standard operation of the the suEXEC wrapper's security model. It is somewhat stringent and can impose new limitations and guidelines for CGI/SSI design, but it was developed carefully step-by-step with security in mind.

For more information as to how this security model can limit your possibilities in regards to server configuration, as well as what security risks can be avoided with a proper suEXEC setup, see the "[Beware the Jabberwock](#)" section of this document.

[BACK TO CONTENTS](#)

Configuring & Installing suEXEC

Here's where we begin the fun. If you use Apache 1.2 or prefer to configure Apache 1.3 with the "src/Configure" script you have to edit the suEXEC header file and install the binary in its proper location manually. This procedure is described in an [extra document](#). The following sections describe the configuration and installation for Apache 1.3 with the AutoConf-style interface (APACI).

APACI's suEXEC configuration options

`--enable-suexec`

This option enables the suEXEC feature which is never installed or activated by default. At least one `--suexec-xxxxx` option has to be provided together with the `--enable-suexec` option to let APACI accept your request for using the suEXEC feature.

`--suexec-caller=UID`

The [username](#) under which Apache normally runs. This is the only user allowed to execute this program.

`--suexec-docroot=DIR`

Define as the DocumentRoot set for Apache. This will be the only hierarchy (aside from UserDirs) that can be used for suEXEC behavior. The default directory is the `--datadir` value with the suffix "/htdocs", *e.g.* if you configure with `--datadir=/home/apache` the directory "/home/apache/htdocs" is used as document root for the suEXEC wrapper.

`--suexec-logfile=FILE`

This defines the filename to which all suEXEC transactions and errors are logged (useful for auditing and debugging purposes). By default the logfile is named "suexec_log" and located in your standard logfile directory (`--logfiledir`).

--suexec-userdir=*DIR*

Define to be the subdirectory under users' home directories where suEXEC access should be allowed. All executables under this directory will be executable by suEXEC as the user so they should be "safe" programs. If you are using a "simple" UserDir directive (ie. one without a "*" in it) this should be set to the same value. suEXEC will not work properly in cases where the UserDir directive points to a location that is not the same as the user's home directory as referenced in the passwd file. Default value is "public_html".

If you have virtual hosts with a different UserDir for each, you will need to define them to all reside in one parent directory; then name that parent directory here. **If this is not defined properly, "~userdir" cgi requests will not work!**

--suexec-uidmin=*UID*

Define this as the lowest UID allowed to be a target user for suEXEC. For most systems, 500 or 100 is common. Default value is 100.

--suexec-gidmin=*GID*

Define this as the lowest GID allowed to be a target group for suEXEC. For most systems, 100 is common and therefore used as default value.

--suexec-safeopath=*PATH*

Define a safe PATH environment to pass to CGI executables. Default value is "/usr/local/bin:/usr/bin:/bin".

Checking your suEXEC setup

Before you compile and install the suEXEC wrapper you can check the configuration with the --layout option.

Example output:

```
suEXEC setup:
  suexec binary: /usr/local/apache/sbin/suexec
  document root: /usr/local/apache/share/htdocs
  userdir suffix: public_html
  logfile: /usr/local/apache/var/log/suexec_log
  safe path: /usr/local/bin:/usr/bin:/bin
  caller ID: www
  minimum user ID: 100
  minimum group ID: 100
```

Compiling and installing the suEXEC wrapper

If you have enabled the suEXEC feature with the --enable-suexec option the suexec binary (together with Apache itself) is automatically built if you execute the command "make".

After all components have been built you can execute the command "make install" to install them. The binary image "suexec" is installed in the directory defined by the --sbindir option. Default location is "/usr/local/apache/sbin/suexec".

Please note that you need **root privileges** for the installation step. In order for the wrapper to set the user ID, it must be installed as owner *root* and must have the setuserid execution bit set for file modes.

[BACK TO CONTENTS](#)

Enabling & Disabling suEXEC

Upon startup of Apache, it looks for the file "suexec" in the "sbin" directory (default is "/usr/local/apache/sbin/suexec"). If Apache finds a properly configured suEXEC wrapper, it will print the following message to the error log:

```
[notice] suEXEC mechanism enabled (wrapper: /path/to/suexec)
```

If you don't see this message at server startup, the server is most likely not finding the wrapper program where it expects it, or the executable is not installed *setuid root*.

If you want to enable the suEXEC mechanism for the first time and an Apache server is already running you must kill and restart Apache. Restarting it with a simple HUP or USR1 signal will not be enough.

If you want to disable suEXEC you should kill and restart Apache after you have removed the "suexec" file.

[BACK TO CONTENTS](#)

Using suEXEC

Virtual Hosts:

One way to use the suEXEC wrapper is through the [User](#) and [Group](#) directives in [VirtualHost](#) definitions. By setting these directives to values different from the main server user ID, all requests for CGI resources will be executed as the *User* and *Group* defined for that <VirtualHost>. If only one or neither of these directives are specified for a <VirtualHost> then the main server userid is assumed.

User directories:

The suEXEC wrapper can also be used to execute CGI programs as the user to which the request is being directed. This is accomplished by using the "~" character prefixing the user ID for whom execution is desired. The only requirement needed for this feature to work is for CGI execution to be enabled for the user and that the script must meet the scrutiny of the [security checks](#) above.

[BACK TO CONTENTS](#)

Debugging suEXEC

The suEXEC wrapper will write log information to the file defined with the --suexec-logfile option as indicated above. If you feel you have configured and installed the wrapper properly, have a look at this log and the error_log for the server to see where you may have gone astray.

[BACK TO CONTENTS](#)

Beware the Jabberwock: Warnings & Examples

NOTE! This section may not be complete. For the latest revision of this section of the documentation, see the Apache Group's [Online Documentation](#) version.

There are a few points of interest regarding the wrapper that can cause limitations on server setup. Please review these before submitting any "bugs" regarding suEXEC.

- **suEXEC Points Of Interest**

- Hierarchy limitations

For security and efficiency reasons, all suexec requests must remain within either a top-level document root for virtual host requests, or one top-level personal document root for userdir requests. For example, if you have four VirtualHosts configured, you would need to structure all of your VHosts' document roots off of one main Apache document hierarchy to take advantage of suEXEC for VirtualHosts. (Example forthcoming.)

- suEXEC's PATH environment variable

This can be a dangerous thing to change. Make certain every path you include in this define is a **trusted** directory. You don't want to open people up to having someone from across the world running a trojan horse on them.

- Altering the suEXEC code

Again, this can cause **Big Trouble** if you try this without knowing what you are doing. Stay away from it if at all possible.

[BACK TO CONTENTS](#)

Apache HTTP Server





Apache HTTP Server Version 1.3

Apache Performance Notes

Author: Dean Gaudet

- [Introduction](#)
 - [Hardware and Operating System Issues](#)
 - [Run-Time Configuration Issues](#)
 - [Compile-Time Configuration Issues](#)
 - Appendixes
 - [Detailed Analysis of a Trace](#)
 - [Patches Available](#)
 - [The Pre-Forking Model](#)
-

Introduction

Apache is a general webserver, which is designed to be correct first, and fast second. Even so, its performance is quite satisfactory. Most sites have less than 10Mbits of outgoing bandwidth, which Apache can fill using only a low end Pentium-based webserver. In practice sites with more bandwidth require more than one machine to fill the bandwidth due to other constraints (such as CGI or database transaction overhead). For these reasons the development focus has been mostly on correctness and configurability.

Unfortunately many folks overlook these facts and cite raw performance numbers as if they are some indication of the quality of a web server product. There is a bare minimum performance that is acceptable, beyond that extra speed only caters to a much smaller segment of the market. But in order to avoid this hurdle to the acceptance of Apache in some markets, effort was put into Apache 1.3 to bring performance up to a point where the difference with other high-end webserver is minimal.

Finally there are the folks who just plain want to see how fast something can go. The author falls into this category. The rest of this document is dedicated to these folks who want to squeeze every last bit of performance out of Apache's current model, and want to understand why it does some things which slow it down.

Note that this is tailored towards Apache 1.3 on Unix. Some of it applies to Apache on NT. Apache on NT has not been tuned for performance yet; in fact it probably performs very poorly because NT performance requires a different programming model.

Hardware and Operating System Issues

The single biggest hardware issue affecting webserver performance is RAM. A webserver should never ever have to swap, swapping increases the latency of each request beyond a point that users consider "fast enough". This causes users to hit stop and reload, further increasing the load. You can, and should, control the `MaxClients` setting so that your server does not spawn so many children it starts swapping.

Beyond that the rest is mundane: get a fast enough CPU, a fast enough network card, and fast enough disks, where "fast enough" is something that needs to be determined by experimentation.

Operating system choice is largely a matter of local concerns. But a general guideline is to always apply the latest vendor TCP/IP patches. HTTP serving completely breaks many of the assumptions built into Unix kernels up through 1994 and even 1995. Good choices include recent FreeBSD, and Linux.

Run-Time Configuration Issues

HostnameLookups

Prior to Apache 1.3, `HostnameLookups` defaulted to `On`. This adds latency to every request because it requires a DNS lookup to complete before the request is finished. In Apache 1.3 this setting defaults to `Off`. However (1.3 or later), if you use any `Allow from domain` or `Deny from domain` directives then you will pay for a double reverse DNS lookup (a reverse, followed by a forward to make sure that the reverse is not being spoofed). So for the highest performance avoid using these directives (it's fine to use IP addresses rather than domain names).

Note that it's possible to scope the directives, such as within a `<Location /server-status>` section. In this case the DNS lookups are only performed on requests matching the criteria. Here's an example which disables lookups except for `.html` and `.cgi` files:

```
HostnameLookups off
<Files ~ "\.(html|cgi)$">
    HostnameLookups on
</Files>
```

But even still, if you just need DNS names in some CGIs you could consider doing the `gethostbyname` call in the specific CGIs that need it.

Similarly, if you need to have hostname information in your server logs in order to generate reports of this information, you can postprocess your log file with [logresolve](#), so that these lookups can be done without making the client wait. It is recommended that you do this postprocessing, and any other statistical analysis of the log file, somewhere other than your production web server machine, in order that this activity does not adversely affect server performance.

FollowSymLinks and SymLinksIfOwnerMatch

Wherever in your URL-space you do not have an `Options FollowSymLinks`, or you do have an `Options SymLinksIfOwnerMatch` Apache will have to issue extra system calls to check up on symlinks. One extra call per filename component. For example, if you had:

```
DocumentRoot /www/htdocs
<Directory />
    Options SymLinksIfOwnerMatch
</Directory>
```

and a request is made for the URI `/index.html`. Then Apache will perform `lstat(2)` on `/www`, `/www/htdocs`, and `/www/htdocs/index.html`. The results of these `lstats` are never cached, so they will occur on every single request. If you really desire the symlinks security checking you can do something like this:

```
DocumentRoot /www/htdocs
<Directory />
    Options FollowSymLinks
</Directory>
<Directory /www/htdocs>
    Options -FollowSymLinks +SymLinksIfOwnerMatch
</Directory>
```

This at least avoids the extra checks for the `DocumentRoot` path. Note that you'll need to add similar sections if you have any `Alias` or `RewriteRule` paths outside of your document root. For highest performance, and no symlink protection, set `FollowSymLinks` everywhere, and never set `SymLinksIfOwnerMatch`.

AllowOverride

Wherever in your URL-space you allow overrides (typically `.htaccess` files) Apache will attempt to open `.htaccess` for each filename component. For example,

```
DocumentRoot /www/htdocs
<Directory />
    AllowOverride all
</Directory>
```

and a request is made for the URI `/index.html`. Then Apache will attempt to open `/.htaccess`, `/www/.htaccess`, and

`/www/htdocs/.htaccess`. The solutions are similar to the previous case of `Options FollowSymLinks`. For highest performance use `AllowOverride None` everywhere in your filesystem.

Negotiation

If at all possible, avoid content-negotiation if you're really interested in every last ounce of performance. In practice the benefits of negotiation outweigh the performance penalties. There's one case where you can speed up the server. Instead of using a wildcard such as:

```
DirectoryIndex index
```

Use a complete list of options:

```
DirectoryIndex index.cgi index.pl index.shtml index.html
```

where you list the most common choice first.

Process Creation

Prior to Apache 1.3 the `MinSpareServers`, `MaxSpareServers`, and `StartServers` settings all had drastic effects on benchmark results. In particular, Apache required a "ramp-up" period in order to reach a number of children sufficient to serve the load being applied. After the initial spawning of `StartServers` children, only one child per second would be created to satisfy the `MinSpareServers` setting. So a server being accessed by 100 simultaneous clients, using the default `StartServers` of 5 would take on the order 95 seconds to spawn enough children to handle the load. This works fine in practice on real-life servers, because they aren't restarted frequently. But does really poorly on benchmarks which might only run for ten minutes.

The one-per-second rule was implemented in an effort to avoid swamping the machine with the startup of new children. If the machine is busy spawning children it can't service requests. But it has such a drastic effect on the perceived performance of Apache that it had to be replaced. As of Apache 1.3, the code will relax the one-per-second rule. It will spawn one, wait a second, then spawn two, wait a second, then spawn four, and it will continue exponentially until it is spawning 32 children per second. It will stop whenever it satisfies the `MinSpareServers` setting.

This appears to be responsive enough that it's almost unnecessary to twiddle the `MinSpareServers`, `MaxSpareServers` and `StartServers` knobs. When more than 4 children are spawned per second, a message will be emitted to the `ErrorLog`. If you see a lot of these errors then consider tuning these settings. Use the `mod_status` output as a guide.

Related to process creation is process death induced by the `MaxRequestsPerChild` setting. By default this is 0, which means that there is no limit to the number of requests handled per child. If your configuration currently has this set to some very low number, such as 30, you may want to bump this up significantly. If you are running SunOS or an old version of Solaris, limit this to 10000 or so because of memory leaks.

When keep-alives are in use, children will be kept busy doing nothing waiting for more requests on the already open connection. The default `KeepAliveTimeout` of 15 seconds attempts to minimize this effect. The tradeoff here is between network bandwidth and server resources. In no event should you raise this above about 60 seconds, as [most of the benefits are lost](#).

Compile-Time Configuration Issues

`mod_status` and `ExtendedStatus On`

If you include `mod_status` and you also set `ExtendedStatus On` when building and running Apache, then on every request Apache will perform two calls to `gettimeofday(2)` (or `times(2)` depending on your operating system), and (pre-1.3) several extra calls to `time(2)`. This is all done so that the status report contains timing indications. For highest performance, set `ExtendedStatus off` (which is the default).

`accept` Serialization - multiple sockets

This discusses a shortcoming in the Unix socket API. Suppose your web server uses multiple `Listen` statements to listen on either multiple ports or multiple addresses. In order to test each socket to see if a connection is ready Apache uses `select(2)`. `select(2)` indicates that a socket has *zero* or *at least one* connection waiting on it. Apache's model includes multiple children, and all the idle ones test for new connections at the same time. A naive implementation looks something like this (these examples do not match the code, they're contrived for pedagogical purposes):

```

for (;;) {
for (;;) {
    fd_set accept_fds;

    FD_ZERO (&accept_fds);
    for (i = first_socket; i <= last_socket; ++i) {
        FD_SET (i, &accept_fds);
    }
    rc = select (last_socket+1, &accept_fds, NULL, NULL, NULL);
    if (rc < 1) continue;
    new_connection = -1;
    for (i = first_socket; i <= last_socket; ++i) {
        if (FD_ISSET (i, &accept_fds)) {
            new_connection = accept (i, NULL, NULL);
            if (new_connection != -1) break;
        }
    }
    if (new_connection != -1) break;
}
process the new_connection;
}

```

But this naive implementation has a serious starvation problem. Recall that multiple children execute this loop at the same time, and so multiple children will block at `select` when they are in between requests. All those blocked children will awaken and return from `select` when a single request appears on any socket (the number of children which awaken varies depending on the operating system and timing issues). They will all then fall down into the loop and try to `accept` the connection. But only one will succeed (assuming there's still only one connection ready), the rest will be *blocked* in `accept`. This effectively locks those children into serving requests from that one socket and no other sockets, and they'll be stuck there until enough new requests appear on that socket to wake them all up. This starvation problem was first documented in [PR#467](#). There are at least two solutions.

One solution is to make the sockets non-blocking. In this case the `accept` won't block the children, and they will be allowed to continue immediately. But this wastes CPU time. Suppose you have ten idle children in `select`, and one connection arrives. Then nine of those children will wake up, try to `accept` the connection, fail, and loop back into `select`, accomplishing nothing. Meanwhile none of those children are servicing requests that occurred on other sockets until they get back up to the `select` again. Overall this solution does not seem very fruitful unless you have as many idle CPUs (in a multiprocessor box) as you have idle children, not a very likely situation.

Another solution, the one used by Apache, is to serialize entry into the inner loop. The loop looks like this (differences highlighted):

```

for (;;) {
accept_mutex_on ();
for (;;) {
    fd_set accept_fds;

    FD_ZERO (&accept_fds);
    for (i = first_socket; i <= last_socket; ++i) {
        FD_SET (i, &accept_fds);
    }
    rc = select (last_socket+1, &accept_fds, NULL, NULL, NULL);
    if (rc < 1) continue;
    new_connection = -1;
    for (i = first_socket; i <= last_socket; ++i) {
        if (FD_ISSET (i, &accept_fds)) {
            new_connection = accept (i, NULL, NULL);
            if (new_connection != -1) break;
        }
    }
    if (new_connection != -1) break;
}
accept_mutex_off ();
process the new_connection;
}

```

The functions `accept_mutex_on` and `accept_mutex_off` implement a mutual exclusion semaphore. Only one child can have the mutex at any time. There are several choices for implementing these mutexes. The choice is defined in `src/conf.h` (pre-1.3) or `src/include/ap_config.h` (1.3 or later). Some architectures do not have any locking choice made, on these architectures it is

unsafe to use multiple `Listen` directives.

`HAVE_FLOCK_SERIALIZED_ACCEPT`

This method uses the `flock(2)` system call to lock a lock file (located by the `LockFile` directive).

`HAVE_FCNTL_SERIALIZED_ACCEPT`

This method uses the `fcntl(2)` system call to lock a lock file (located by the `LockFile` directive).

`HAVE_SYSVSEM_SERIALIZED_ACCEPT`

(1.3 or later) This method uses SysV-style semaphores to implement the mutex. Unfortunately SysV-style semaphores have some bad side-effects. One is that it's possible Apache will die without cleaning up the semaphore (see the `ipcs(8)` man page). The other is that the semaphore API allows for a denial of service attack by any CGIs running under the same uid as the webserver (*i.e.*, all CGIs, unless you use something like `suexec` or `cgiwrapper`). For these reasons this method is not used on any architecture except IRIX (where the previous two are prohibitively expensive on most IRIX boxes).

`HAVE_USLOCK_SERIALIZED_ACCEPT`

(1.3 or later) This method is only available on IRIX, and uses `usconfig(2)` to create a mutex. While this method avoids the hassles of SysV-style semaphores, it is not the default for IRIX. This is because on single processor IRIX boxes (5.3 or 6.2) the `uslock` code is two orders of magnitude slower than the SysV-semaphore code. On multi-processor IRIX boxes the `uslock` code is an order of magnitude faster than the SysV-semaphore code. Kind of a messed up situation. So if you're using a multiprocessor IRIX box then you should rebuild your webserver with `-DHAVE_USLOCK_SERIALIZED_ACCEPT` on the `EXTRA_CFLAGS`.

`HAVE_PTHREAD_SERIALIZED_ACCEPT`

(1.3 or later) This method uses POSIX mutexes and should work on any architecture implementing the full POSIX threads specification, however appears to only work on Solaris (2.5 or later), and even then only in certain configurations. If you experiment with this you should watch out for your server hanging and not responding. Static content only servers may work just fine.

If your system has another method of serialization which isn't in the above list then it may be worthwhile adding code for it (and submitting a patch back to Apache). The above `HAVE_METHOD_SERIALIZED_ACCEPT` defines specify which method is available and works on the platform (you can have more than one); `USE_METHOD_SERIALIZED_ACCEPT` is used to specify the default method (see the `AcceptMutex` directive).

Another solution that has been considered but never implemented is to partially serialize the loop -- that is, let in a certain number of processes. This would only be of interest on multiprocessor boxes where it's possible multiple children could run simultaneously, and the serialization actually doesn't take advantage of the full bandwidth. This is a possible area of future investigation, but priority remains low because highly parallel web servers are not the norm.

Ideally you should run servers without multiple `Listen` statements if you want the highest performance. But read on.

accept Serialization - single socket

The above is fine and dandy for multiple socket servers, but what about single socket servers? In theory they shouldn't experience any of these same problems because all children can just block in `accept(2)` until a connection arrives, and no starvation results. In practice this hides almost the same "spinning" behavior discussed above in the non-blocking solution. The way that most TCP stacks are implemented, the kernel actually wakes up all processes blocked in `accept` when a single connection arrives. One of those processes gets the connection and returns to user-space, the rest spin in the kernel and go back to sleep when they discover there's no connection for them. This spinning is hidden from the user-land code, but it's there nonetheless. This can result in the same load-spiking wasteful behavior that a non-blocking solution to the multiple sockets case can.

For this reason we have found that many architectures behave more "nicely" if we serialize even the single socket case. So this is actually the default in almost all cases. Crude experiments under Linux (2.0.30 on a dual Pentium pro 166 w/128Mb RAM) have shown that the serialization of the single socket case causes less than a 3% decrease in requests per second over unserialized single-socket. But unserialized single-socket showed an extra 100ms latency on each request. This latency is probably a wash on long haul lines, and only an issue on LANs. If you want to override the single socket serialization you can define `SINGLE_LISTEN_UNSERIALIZED_ACCEPT` and then single-socket servers will not serialize at all.

Lingering Close

As discussed in [draft-ietf-http-connection-00.txt](#) section 8, in order for an HTTP server to **reliably** implement the protocol it needs to shutdown each direction of the communication independently (recall that a TCP connection is bi-directional, each half is independent of the other). This fact is often overlooked by other servers, but is correctly implemented in Apache as of 1.2.

When this feature was added to Apache it caused a flurry of problems on various versions of Unix because of a shortsightedness. The TCP specification does not state that the `FIN_WAIT_2` state has a timeout, but it doesn't prohibit it. On systems without the timeout,

Apache 1.2 induces many sockets stuck forever in the FIN_WAIT_2 state. In many cases this can be avoided by simply upgrading to the latest TCP/IP patches supplied by the vendor. In cases where the vendor has never released patches (*i.e.*, SunOS4 -- although folks with a source license can patch it themselves) we have decided to disable this feature.

There are two ways of accomplishing this. One is the socket option SO_LINGER. But as fate would have it, this has never been implemented properly in most TCP/IP stacks. Even on those stacks with a proper implementation (*i.e.*, Linux 2.0.31) this method proves to be more expensive (cputime) than the next solution.

For the most part, Apache implements this in a function called `lingering_close` (in `http_main.c`). The function looks roughly like this:

```
void lingering_close (int s)
{
    char junk_buffer[2048];

    /* shutdown the sending side */
    shutdown (s, 1);

    signal (SIGALRM, lingering_death);
    alarm (30);

    for (;;) {
        select (s for reading, 2 second timeout);
        if (error) break;
        if (s is ready for reading) {
            if (read (s, junk_buffer, sizeof (junk_buffer)) <= 0) {
                break;
            }
            /* just toss away whatever is read */
        }
    }

    close (s);
}
```

This naturally adds some expense at the end of a connection, but it is required for a reliable implementation. As HTTP/1.1 becomes more prevalent, and all connections are persistent, this expense will be amortized over more requests. If you want to play with fire and disable this feature you can define `NO_LINGCLOSE`, but this is not recommended at all. In particular, as HTTP/1.1 pipelined persistent connections come into use `lingering_close` is an absolute necessity (and [pipelined connections are faster](#), so you want to support them).

Scoreboard File

Apache's parent and children communicate with each other through something called the scoreboard. Ideally this should be implemented in shared memory. For those operating systems that we either have access to, or have been given detailed ports for, it typically is implemented using shared memory. The rest default to using an on-disk file. The on-disk file is not only slow, but it is unreliable (and less featured). Peruse the `src/main/conf.h` file for your architecture and look for either `USE_MMAP_SCOREBOARD` or `USE_SHMGET_SCOREBOARD`. Defining one of those two (as well as their companions `HAVE_MMAP` and `HAVE_SHMGET` respectively) enables the supplied shared memory code. If your system has another type of shared memory, edit the file `src/main/http_main.c` and add the hooks necessary to use it in Apache. (Send us back a patch too please.)

Historical note: The Linux port of Apache didn't start to use shared memory until version 1.2 of Apache. This oversight resulted in really poor and unreliable behavior of earlier versions of Apache on Linux.

DYNAMIC_MODULE_LIMIT

If you have no intention of using dynamically loaded modules (you probably don't if you're reading this and tuning your server for every last ounce of performance) then you should add `-DDYNAMIC_MODULE_LIMIT=0` when building your server. This will save RAM that's allocated only for supporting dynamically loaded modules.

Appendix: Detailed Analysis of a Trace

Here is a system call trace of Apache 1.3 running on Linux. The run-time configuration file is essentially the default plus:

```
<Directory />
    AllowOverride none
    Options FollowSymLinks
</Directory>
```

The file being requested is a static 6K file of no particular content. Traces of non-static requests or requests with content negotiation look wildly different (and quite ugly in some cases). First the entire trace, then we'll examine details. (This was generated by the `strace` program, other similar programs include `truss`, `ktrace`, and `par`.)

```
accept(15, {sin_family=AF_INET, sin_port=htons(22283),
sin_addr=inet_addr("127.0.0.1")}, [16]) = 3
flock(18, LOCK_UN) = 0
sigaction(SIGUSR1, {SIG_IGN}, {0x8059954, [], SA_INTERRUPT}) = 0
getsockname(3, {sin_family=AF_INET, sin_port=htons(8080),
sin_addr=inet_addr("127.0.0.1")}, [16]) = 0
setsockopt(3, IPPROTO_TCP, [1], 4) = 0
read(3, "GET /6k HTTP/1.0\r\nUser-Agent: "..., 4096) = 60
sigaction(SIGUSR1, {SIG_IGN}, {SIG_IGN}) = 0
time(NULL) = 873959960
gettimeofday({873959960, 404935}, NULL) = 0
stat("/home/dgaudet/ap/apachen/htdocs/6k", {st_mode=S_IFREG|0644, st_size=6144, ...}) =
0
open("/home/dgaudet/ap/apachen/htdocs/6k", O_RDONLY) = 4
mmap(0, 6144, PROT_READ, MAP_PRIVATE, 4, 0) = 0x400ee000
writev(3, [{"HTTP/1.1 200 OK\r\nDate: Thu, 11"..., 245},
{"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"..., 6144}], 2) = 6389
close(4) = 0
time(NULL) = 873959960
write(17, "127.0.0.1 - - [10/Sep/1997:23:39"..., 71) = 71
gettimeofday({873959960, 417742}, NULL) = 0
times({tms_utime=5, tms_stime=0, tms_cutime=0, tms_cstime=0}) = 446747
shutdown(3, 1 /* send */) = 0
oldselect(4, [3], NULL, [3], {2, 0}) = 1 (in [3], left {2, 0})
read(3, "", 2048) = 0
close(3) = 0
sigaction(SIGUSR1, {0x8059954, [], SA_INTERRUPT}, {SIG_IGN}) = 0
munmap(0x400ee000, 6144) = 0
flock(18, LOCK_EX) = 0
```

Notice the accept serialization:

```
flock(18, LOCK_UN) = 0
...
flock(18, LOCK_EX) = 0
```

These two calls can be removed by defining `SINGLE_LISTEN_UNSERIALIZED_ACCEPT` as described earlier.

Notice the SIGUSR1 manipulation:

```
sigaction(SIGUSR1, {SIG_IGN}, {0x8059954, [], SA_INTERRUPT}) = 0
...
sigaction(SIGUSR1, {SIG_IGN}, {SIG_IGN}) = 0
...
sigaction(SIGUSR1, {0x8059954, [], SA_INTERRUPT}, {SIG_IGN}) = 0
```

This is caused by the implementation of graceful restarts. When the parent receives a SIGUSR1 it sends a SIGUSR1 to all of its children (and it also increments a "generation counter" in shared memory). Any children that are idle (between connections) will immediately die off when they receive the signal. Any children that are in keep-alive connections, but are in between requests will die off immediately. But any children that have a connection and are still waiting for the first request will not die off immediately.

To see why this is necessary, consider how a browser reacts to a closed connection. If the connection was a keep-alive connection and the

request being serviced was not the first request then the browser will quietly reissue the request on a new connection. It has to do this because the server is always free to close a keep-alive connection in between requests (*i.e.*, due to a timeout or because of a maximum number of requests). But, if the connection is closed before the first response has been received the typical browser will display a "document contains no data" dialogue (or a broken image icon). This is done on the assumption that the server is broken in some way (or maybe too overloaded to respond at all). So Apache tries to avoid ever deliberately closing the connection before it has sent a single response. This is the cause of those SIGUSR1 manipulations.

Note that it is theoretically possible to eliminate all three of these calls. But in rough tests the gain proved to be almost unnoticeable.

In order to implement virtual hosts, Apache needs to know the local socket address used to accept the connection:

```
getsockname(3, {sin_family=AF_INET, sin_port=htons(8080),
sin_addr=inet_addr("127.0.0.1")}, [16]) = 0
```

It is possible to eliminate this call in many situations (such as when there are no virtual hosts, or when Listen directives are used which do not have wildcard addresses). But no effort has yet been made to do these optimizations.

Apache turns off the Nagle algorithm:

```
setsockopt(3, IPPROTO_TCP, [1], 4) = 0
```

because of problems described in [a paper by John Heidemann](#).

Notice the two time calls:

```
time(NULL) = 873959960
...
time(NULL) = 873959960
```

One of these occurs at the beginning of the request, and the other occurs as a result of writing the log. At least one of these is required to properly implement the HTTP protocol. The second occurs because the Common Log Format dictates that the log record include a timestamp of the end of the request. A custom logging module could eliminate one of the calls. Or you can use a method which moves the time into shared memory, see the [patches section below](#).

As described earlier, ExtendedStatus On causes two gettimeofday calls and a call to times:

```
gettimeofday({873959960, 404935}, NULL) = 0
...
gettimeofday({873959960, 417742}, NULL) = 0
times({tms_utime=5, tms_stime=0, tms_cutime=0, tms_cstime=0}) = 446747
```

These can be removed by setting ExtendedStatus Off (which is the default).

It might seem odd to call stat:

```
stat("/home/dgaudet/ap/apachen/htdocs/6k", {st_mode=S_IFREG|0644, st_size=6144, ...}) = 0
```

This is part of the algorithm which calculates the PATH_INFO for use by CGIs. In fact if the request had been for the URI /cgi-bin/printenv/foobar then there would be two calls to stat. The first for /home/dgaudet/ap/apachen/cgi-bin/printenv/foobar which does not exist, and the second for /home/dgaudet/ap/apachen/cgi-bin/printenv, which does exist. Regardless, at least one stat call is necessary when serving static files because the file size and modification times are used to generate HTTP headers (such as Content-Length, Last-Modified) and implement protocol features (such as If-Modified-Since). A somewhat more clever server could avoid the stat when serving non-static files, however doing so in Apache is very difficult given the modular structure.

All static files are served using mmap:

```
mmap(0, 6144, PROT_READ, MAP_PRIVATE, 4, 0) = 0x400ee000
...
munmap(0x400ee000, 6144) = 0
```

On some architectures it's slower to mmap small files than it is to simply read them. The define MMAP_THRESHOLD can be set to the minimum size required before using mmap. By default it's set to 0 (except on SunOS4 where experimentation has shown 8192 to be a better value). Using a tool such as [lmbench](#) you can determine the optimal setting for your environment.

You may also wish to experiment with `MMAP_SEGMENT_SIZE` (default 32768) which determines the maximum number of bytes that will be written at a time from `mmap()`d files. Apache only resets the client's `Timeout` in between `write()`s. So setting this large may lock out low bandwidth clients unless you also increase the `Timeout`.

It may even be the case that `mmap` isn't used on your architecture; if so then defining `USE_MMAP_FILES` and `HAVE_MMAP` might work (if it works then report back to us).

Apache does its best to avoid copying bytes around in memory. The first write of any request typically is turned into a `writtev` which combines both the headers and the first hunk of data:

```
writtev(3, [{"HTTP/1.1 200 OK\r\nDate: Thu, 11"... , 245},
{"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"... , 6144}], 2) = 6389
```

When doing HTTP/1.1 chunked encoding Apache will generate up to four element `writtevs`. The goal is to push the byte copying into the kernel, where it typically has to happen anyhow (to assemble network packets). On testing, various Unixes (BSDI 2.x, Solaris 2.5, Linux 2.0.31+) properly combine the elements into network packets. Pre-2.0.31 Linux will not combine, and will create a packet for each element, so upgrading is a good idea. Defining `NO_WRITEV` will disable this combining, but result in very poor chunked encoding performance.

The log write:

```
write(17, "127.0.0.1 - - [10/Sep/1997:23:39"... , 71) = 71
```

can be deferred by defining `BUFFERED_LOGS`. In this case up to `PIPE_BUF` bytes (a POSIX defined constant) of log entries are buffered before writing. At no time does it split a log entry across a `PIPE_BUF` boundary because those writes may not be atomic. (*i.e.*, entries from multiple children could become mixed together). The code does its best to flush this buffer when a child dies.

The lingering close code causes four system calls:

```
shutdown(3, 1 /* send */) = 0
oldselect(4, [3], NULL, [3], {2, 0}) = 1 (in [3], left {2, 0})
read(3, "", 2048) = 0
close(3) = 0
```

which were described earlier.

Let's apply some of these optimizations: `-DSINGLE_LISTEN_UNSERIALIZED_ACCEPT` `-DBUFFERED_LOGS` and `ExtendedStatus Off`. Here's the final trace:

```
accept(15, {sin_family=AF_INET, sin_port=htons(22286),
sin_addr=inet_addr("127.0.0.1")}, [16]) = 3
sigaction(SIGUSR1, {SIG_IGN}, {0x8058c98, [], SA_INTERRUPT}) = 0
getsockname(3, {sin_family=AF_INET, sin_port=htons(8080),
sin_addr=inet_addr("127.0.0.1")}, [16]) = 0
setsockopt(3, IPPROTO_TCP, [1], 4) = 0
read(3, "GET /6k HTTP/1.0\r\nUser-Agent: "... , 4096) = 60
sigaction(SIGUSR1, {SIG_IGN}, {SIG_IGN}) = 0
time(NULL) = 873961916
stat("/home/dgaudet/ap/apachen/htdocs/6k", {st_mode=S_IFREG|0644, st_size=6144, ...}) =
0
open("/home/dgaudet/ap/apachen/htdocs/6k", O_RDONLY) = 4
mmap(0, 6144, PROT_READ, MAP_PRIVATE, 4, 0) = 0x400e3000
writtev(3, [{"HTTP/1.1 200 OK\r\nDate: Thu, 11"... , 245},
{"\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0"... , 6144}], 2) = 6389
close(4) = 0
time(NULL) = 873961916
shutdown(3, 1 /* send */) = 0
oldselect(4, [3], NULL, [3], {2, 0}) = 1 (in [3], left {2, 0})
read(3, "", 2048) = 0
close(3) = 0
sigaction(SIGUSR1, {0x8058c98, [], SA_INTERRUPT}, {SIG_IGN}) = 0
munmap(0x400e3000, 6144) = 0
```

That's 19 system calls, of which 4 remain relatively easy to remove, but don't seem worth the effort.

Appendix: Patches Available

There are [several performance patches available for 1.3](#). Although they may not apply cleanly to the current version, it shouldn't be difficult for someone with a little C knowledge to update them. In particular:

- A [patch](#) to remove all `time(2)` system calls.
- A [patch](#) to remove various system calls from `mod_include`, these calls are used by few sites but required for backwards compatibility.
- A [patch](#) which integrates the above two plus a few other speedups at the cost of removing some functionality.

Appendix: The Pre-Forking Model

Apache (on Unix) is a *pre-forking* model server. The *parent* process is responsible only for forking *child* processes, it does not serve any requests or service any network sockets. The child processes actually process connections, they serve multiple connections (one at a time) before dying. The parent spawns new or kills off old children in response to changes in the load on the server (it does so by monitoring a scoreboard which the children keep up to date).

This model for servers offers a robustness that other models do not. In particular, the parent code is very simple, and with a high degree of confidence the parent will continue to do its job without error. The children are complex, and when you add in third party code via modules, you risk segmentation faults and other forms of corruption. Even should such a thing happen, it only affects one connection and the server continues serving requests. The parent quickly replaces the dead child.

Pre-forking is also very portable across dialects of Unix. Historically this has been an important goal for Apache, and it continues to remain so.

The pre-forking model comes under criticism for various performance aspects. Of particular concern are the overhead of forking a process, the overhead of context switches between processes, and the memory overhead of having multiple processes. Furthermore it does not offer as many opportunities for data-caching between requests (such as a pool of `mmap`d files). Various other models exist and extensive analysis can be found in the [papers of the JAWS project](#). In practice all of these costs vary drastically depending on the operating system.

Apache's core code is already multithread aware, and Apache version 1.3 is multithreaded on NT. There have been at least two other experimental implementations of threaded Apache, one using the 1.3 code base on DCE, and one using a custom user-level threads package and the 1.0 code base; neither is publicly available. There is also an experimental port of Apache 1.3 to [Netscape's Portable Run Time](#), which [is available](#) (but you're encouraged to join the [new-httpd mailing list](#) if you intend to use it). Part of our redesign for version 2.0 of Apache will include abstractions of the server model so that we can continue to support the pre-forking model, and also support various threaded models.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache 1.3 URL Rewriting Guide

Originally written by
Ralf S. Engelschall <rse@apache.org>
December 1997

This document supplements the mod_rewrite [reference documentation](#). It describes how one can use Apache's mod_rewrite to solve typical URL-based problems webmasters are usually confronted with in practice. I give detailed descriptions on how to solve each problem by configuring URL rewriting rulesets.

Introduction to mod_rewrite

The Apache module mod_rewrite is a killer one, i.e. it is a really sophisticated module which provides a powerful way to do URL manipulations. With it you can nearly do all types of URL manipulations you ever dreamed about. The price you have to pay is to accept complexity, because mod_rewrite's major drawback is that it is not easy to understand and use for the beginner. And even Apache experts sometimes discover new aspects where mod_rewrite can help.

In other words: With mod_rewrite you either shoot yourself in the foot the first time and never use it again or love it for the rest of your life because of its power. This paper tries to give you a few initial success events to avoid the first case by presenting already invented solutions to you.

Practical Solutions

Here come a lot of practical solutions I've either invented myself or collected from other peoples solutions in the past. Feel free to learn the black magic of URL rewriting from these examples.

ATTENTION: Depending on your server-configuration it can be necessary to slightly change the examples for your situation, e.g. adding the [PT] flag when additionally using mod_alias and mod_userdir, etc. Or rewriting a ruleset to fit in .htaccess context instead of per-server context. Always try to understand what a particular ruleset really does before you use it. It avoid problems.

URL Layout

Canonical URLs

Description:

On some webservers there are more than one URL for a resource. Usually there are canonical URLs (which should be actually used and distributed) and those which are just shortcuts, internal ones, etc. Independent which URL the user supplied with the request he should finally see the canonical one only.

Solution:

We do an external HTTP redirect for all non-canonical URLs to fix them in the location view of the Browser and for all subsequent requests. In the example ruleset below we replace /~user by the canonical /u/user and fix a missing trailing slash for /u/user.

```
RewriteRule ^/~([\^/]+)/?(.*) /u/$1/$2 [R]
RewriteRule ^/([uge])/([\^/]+)$ /$1/$2/ [R]
```

Canonical Hostnames

Description:

...

Solution:

```

RewriteCond %{HTTP_HOST} !^fully\.qualified\.domain\.name [NC]
RewriteCond %{HTTP_HOST} !^$
RewriteCond %{SERVER_PORT} !^80$
RewriteRule ^/(.*) http://fully.qualified.domain.name:%{SERVER_PORT}/$1 [L,R]
RewriteCond %{HTTP_HOST} !^fully\.qualified\.domain\.name [NC]
RewriteCond %{HTTP_HOST} !^$
RewriteRule ^/(.*) http://fully.qualified.domain.name/$1 [L,R]

```

Moved DocumentRoot

Description:

Usually the DocumentRoot of the webserver directly relates to the URL ``/". But often this data is not really of top-level priority, it is perhaps just one entity of a lot of data pools. For instance at our Intranet sites there are /e/www/ (the homepage for WWW), /e/sww/ (the homepage for the Intranet) etc. Now because the data of the DocumentRoot stays at /e/www/ we had to make sure that all inlined images and other stuff inside this data pool work for subsequent requests.

Solution:

We just redirect the URL / to /e/www/. While it seems trivial it is actually trivial with mod_rewrite, only. Because the typical old mechanisms of URL *Aliases* (as provided by mod_alias and friends) only used *prefix* matching. With this you cannot do such a redirection because the DocumentRoot is a prefix of all URLs. With mod_rewrite it is really trivial:

```

RewriteEngine on
RewriteRule ^/$ /e/www/ [R]

```

Trailing Slash Problem

Description:

Every webmaster can sing a song about the problem of the trailing slash on URLs referencing directories. If they are missing, the server dumps an error, because if you say /~quux/foo instead of /~quux/foo/ then the server searches for a *file* named foo. And because this file is a directory it complains. Actually it tries to fix it itself in most of the cases, but sometimes this mechanism needs to be emulated by you. For instance after you have done a lot of complicated URL rewritings to CGI scripts etc.

Solution:

The solution to this subtle problem is to let the server add the trailing slash automatically. To do this correctly we have to use an external redirect, so the browser correctly requests subsequent images etc. If we only did an internal rewrite, this would only work for the directory page, but would go wrong when any images are included into this page with relative URLs, because the browser would request an in-lined object. For instance, a request for image.gif in /~quux/foo/index.html would become /~quux/image.gif without the external redirect!

So, to do this trick we write:

```

RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo$ foo/ [R]

```

The crazy and lazy can even do the following in the top-level .htaccess file of their homedir. But notice that this creates some processing overhead.

```

RewriteEngine on
RewriteBase /~quux/
RewriteCond %{REQUEST_FILENAME} -d
RewriteRule ^(.+[^/] )$ $1/ [R]

```

Webcluster through Homogeneous URL Layout

Description:

We want to create a homogenous and consistent URL layout over all WWW servers on a Intranet webcluster, i.e. all URLs (per definition server local and thus server dependent!) become actually server *independent!* What we want is to give the WWW namespace a consistent server-independent layout: no URL should have to include any physically correct target server. The cluster itself should drive us automatically to the physical target host.

Solution:

First, the knowledge of the target servers come from (distributed) external maps which contain information where our users, groups and entities stay. They have the form

```

user1 server_of_user1
user2 server_of_user2
:      :

```

We put them into files `map.xxx-to-host`. Second we need to instruct all servers to redirect URLs of the forms

```

/u/user/anypath
/g/group/anypath
/e/entity/anypath

```

to

```

http://physical-host/u/user/anypath
http://physical-host/g/group/anypath
http://physical-host/e/entity/anypath

```

when the URL is not locally valid to a server. The following ruleset does this for us by the help of the map files (assuming that `server0` is a default server which will be used if a user has no entry in the map):

```

RewriteEngine on

RewriteMap user-to-host txt:/path/to/map.user-to-host
RewriteMap group-to-host txt:/path/to/map.group-to-host
RewriteMap entity-to-host txt:/path/to/map.entity-to-host

RewriteRule ^/u/([^/]+)/?(.*) http://${user-to-host:$1|server0}/u/$1/$2
RewriteRule ^/g/([^/]+)/?(.*) http://${group-to-host:$1|server0}/g/$1/$2
RewriteRule ^/e/([^/]+)/?(.*) http://${entity-to-host:$1|server0}/e/$1/$2

RewriteRule ^/([uge])/([^/]+)/?$ /$1/$2/.www/
RewriteRule ^/([uge])/([^/]+)/([^.]+) /$1/$2/.www/$3\

```

Move Homedirs to Different Webserver

Description:

A lot of webmaster asked for a solution to the following situation: They wanted to redirect just all homedirs on a webserver to another webserver. They usually need such things when establishing a newer webserver which will replace the old one over time.

Solution:

The solution is trivial with `mod_rewrite`. On the old webserver we just redirect all `/~user/anypath` URLs to `http://newsserver/~user/anypath`.

```
RewriteEngine on
RewriteRule ^/~(.+) http://newsserver/~$1 [R,L]
```

Structured Homedirs

Description:

Some sites with thousand of users usually use a structured homedir layout, i.e. each homedir is in a subdirectory which begins for instance with the first character of the username. So, /~foo/anypath is /home/**f**/foo/.www/anypath while /~bar/anypath is /home/**b**/bar/.www/anypath.

Solution:

We use the following ruleset to expand the tilde URLs into exactly the above layout.

```
RewriteEngine on
RewriteRule ^/~(([a-z])[a-z0-9]+)(.*) /home/$2/$1/.www$3
```

Filesystem Reorganisation

Description:

This really is a hardcore example: a killer application which heavily uses per-directory RewriteRules to get a smooth look and feel on the Web while its data structure is never touched or adjusted. Background: *net.sw* is my archive of freely available Unix software packages, which I started to collect in 1992. It is both my hobby and job to do this, because while I'm studying computer science I have also worked for many years as a system and network administrator in my spare time. Every week I need some sort of software so I created a deep hierarchy of directories where I stored the packages:

```
drwxrwxr-x  2 netsw  users      512 Aug  3 18:39 Audio/
drwxrwxr-x  2 netsw  users      512 Jul  9 14:37 Benchmark/
drwxrwxr-x 12 netsw  users      512 Jul  9 00:34 Crypto/
drwxrwxr-x  5 netsw  users      512 Jul  9 00:41 Database/
drwxrwxr-x  4 netsw  users      512 Jul 30 19:25 Dicts/
drwxrwxr-x 10 netsw  users      512 Jul  9 01:54 Graphic/
drwxrwxr-x  5 netsw  users      512 Jul  9 01:58 Hackers/
drwxrwxr-x  8 netsw  users      512 Jul  9 03:19 InfoSys/
drwxrwxr-x  3 netsw  users      512 Jul  9 03:21 Math/
drwxrwxr-x  3 netsw  users      512 Jul  9 03:24 Misc/
drwxrwxr-x  9 netsw  users      512 Aug  1 16:33 Network/
drwxrwxr-x  2 netsw  users      512 Jul  9 05:53 Office/
drwxrwxr-x  7 netsw  users      512 Jul  9 09:24 SoftEng/
drwxrwxr-x  7 netsw  users      512 Jul  9 12:17 System/
drwxrwxr-x 12 netsw  users      512 Aug  3 20:15 Typesetting/
drwxrwxr-x 10 netsw  users      512 Jul  9 14:08 X11/
```

In July 1996 I decided to make this archive public to the world via a nice Web interface. "Nice" means that I wanted to offer an interface where you can browse directly through the archive hierarchy. And "nice" means that I didn't want to change anything inside this hierarchy - not even by putting some CGI scripts at the top of it. Why? Because the above structure should be later accessible via FTP as well, and I didn't want any Web or CGI stuff to be there.

Solution:

The solution has two parts: The first is a set of CGI scripts which create all the pages at all directory levels on-the-fly. I put them under /e/netsw/.www/ as follows:

```
-rw-r--r--  1 netsw  users      1318 Aug  1 18:10 .wwwacl
drwxr-xr-x 18 netsw  users        512 Aug  5 15:51 DATA/
-rw-rw-rw-  1 netsw  users    372982 Aug  5 16:35 LOGFILE
-rw-r--r--  1 netsw  users        659 Aug  4 09:27 TODO
-rw-r--r--  1 netsw  users     5697 Aug  1 18:01 netsw-about.html
-rwxr-xr-x  1 netsw  users        579 Aug  2 10:33 netsw-access.pl
-rwxr-xr-x  1 netsw  users     1532 Aug  1 17:35 netsw-changes.cgi
-rwxr-xr-x  1 netsw  users     2866 Aug  5 14:49 netsw-home.cgi
drwxr-xr-x  2 netsw  users        512 Jul  8 23:47 netsw-img/
```

```

-rwxr-xr-x  1 netsw  users    24050 Aug  5 15:49 netsw-lsdir.cgi
-rwxr-xr-x  1 netsw  users    1589  Aug  3 18:43 netsw-search.cgi
-rwxr-xr-x  1 netsw  users    1885  Aug  1 17:41 netsw-tree.cgi
-rw-r--r--  1 netsw  users     234  Jul 30 16:35 netsw-unlimit.lst

```

The DATA/ subdirectory holds the above directory structure, i.e. the real *net.sw* stuff and gets automatically updated via *rdist* from time to time. The second part of the problem remains: how to link these two structures together into one smooth-looking URL tree? We want to hide the DATA/ directory from the user while running the appropriate CGI scripts for the various URLs. Here is the solution: first I put the following into the per-directory configuration file in the Document Root of the server to rewrite the announced URL `/net.sw/` to the internal path `/e/netsw/`:

```

RewriteRule ^net.sw$      net.sw/      [R]
RewriteRule ^net.sw/(.*)$ e/netsw/$1

```

The first rule is for requests which miss the trailing slash! The second rule does the real thing. And then comes the killer configuration which stays in the per-directory config file `/e/netsw/.www/.wwwacl`:

```

Options          ExecCGI FollowSymLinks Includes MultiViews

RewriteEngine on

# we are reached via /net.sw/ prefix
RewriteBase     /net.sw/

# first we rewrite the root dir to
# the handling cgi script
RewriteRule    ^$                netsw-home.cgi      [L]
RewriteRule    ^index\.html$     netsw-home.cgi      [L]

# strip out the subdirs when
# the browser requests us from perdir pages
RewriteRule    ^.+/(netsw-[^/]+/+.+)$ $1                    [L]

# and now break the rewriting for local files
RewriteRule    ^netsw-home\.cgi.* -                        [L]
RewriteRule    ^netsw-changes\.cgi.* -                       [L]
RewriteRule    ^netsw-search\.cgi.* -                       [L]
RewriteRule    ^netsw-tree\.cgi$ -                            [L]
RewriteRule    ^netsw-about\.html$ -                          [L]
RewriteRule    ^netsw-img/.*$ -                               [L]

# anything else is a subdir which gets handled
# by another cgi script
RewriteRule    !^netsw-lsdir\.cgi.* -                          [C]
RewriteRule    (.*)              netsw-lsdir.cgi/$1

```

Some hints for interpretation:

1. Notice the L (last) flag and no substitution field ('-') in the forth part
2. Notice the ! (not) character and the C (chain) flag at the first rule in the last part
3. Notice the catch-all pattern in the last rule

NCSA imagemap to Apache mod_imap

Description:

When switching from the NCSA webserver to the more modern Apache webserver a lot of people want a smooth transition. So they want pages which use their old NCSA *imagemap* program to work under Apache with the modern *mod_imap*. The problem is that there are a lot of hyperlinks around which reference the *imagemap* program via `/cgi-bin/imagemap/path/to/page.map`. Under Apache this has to read just `/path/to/page.map`.

Solution:

We use a global rule to remove the prefix on-the-fly for all requests:

```
RewriteEngine on
RewriteRule ^/cgi-bin/imagemap(.*) $1 [PT]
```

Search pages in more than one directory

Description:

Sometimes it is necessary to let the webserver search for pages in more than one directory. Here MultiViews or other techniques cannot help.

Solution:

We program a explicit ruleset which searches for the files in the directories.

```
RewriteEngine on

# first try to find it in custom/...
# ...and if found stop and be happy:
RewriteCond /your/docroot/dir1/{REQUEST_FILENAME} -f
RewriteRule ^(.+) /your/docroot/dir1/$1 [L]

# second try to find it in pub/...
# ...and if found stop and be happy:
RewriteCond /your/docroot/dir2/{REQUEST_FILENAME} -f
RewriteRule ^(.+) /your/docroot/dir2/$1 [L]

# else go on for other Alias or ScriptAlias directives,
# etc.
RewriteRule ^(.+) - [PT]
```

Set Environment Variables According To URL Parts

Description:

Perhaps you want to keep status information between requests and use the URL to encode it. But you don't want to use a CGI wrapper for all pages just to strip out this information.

Solution:

We use a rewrite rule to strip out the status information and remember it via an environment variable which can be later dereferenced from within XSSI or CGI. This way a URL `/foo/S=java/bar/` gets translated to `/foo/bar/` and the environment variable named `STATUS` is set to the value "java".

```
RewriteEngine on
RewriteRule ^(.*)/S=([^/]+)/(.*) $1/$3 [E=STATUS:$2]
```

Virtual User Hosts

Description:

Assume that you want to provide `www.username.host.domain.com` for the homepage of username via just DNS A records to the same machine and without any virtualhosts on this machine.

Solution:

For HTTP/1.0 requests there is no solution, but for HTTP/1.1 requests which contain a Host: HTTP header we can use the following ruleset to rewrite `http://www.username.host.com/anypath` internally to `/home/username/anypath`:

```
RewriteEngine on
RewriteCond %{HTTP_HOST} ^www\.[^.]+\\.host\.com$
RewriteRule ^(.+) %{HTTP_HOST}$1 [C]
RewriteRule ^www\.[^.]+\\.host\.com(.*) /home/$1$2
```

Redirect Homedirs For Foreigners

Description:

We want to redirect homedir URLs to another webserver `www.somewhere.com` when the requesting user does not stay in the local domain `ourdomain.com`. This is sometimes used in virtual host contexts.

Solution:

Just a rewrite condition:

```
RewriteEngine on
RewriteCond  %{REMOTE_HOST}    !^\.+\.ourdomain\.com$
RewriteRule  ^(/~.+ )          http://www.somewhere.com/$1 [R,L]
```

Redirect Failing URLs To Other Webserver

Description:

A typical FAQ about URL rewriting is how to redirect failing requests on webserver A to webserver B. Usually this is done via `ErrorDocument` CGI-scripts in Perl, but there is also a `mod_rewrite` solution. But notice that this is less performant than using a `ErrorDocument` CGI-script!

Solution:

The first solution has the best performance but less flexibility and is less error safe:

```
RewriteEngine on
RewriteCond  /your/docroot/%{REQUEST_FILENAME}  !-f
RewriteRule  ^(.+)                               http://webserverB.dom/$1
```

The problem here is that this will only work for pages inside the `DocumentRoot`. While you can add more Conditions (for instance to also handle homedirs, etc.) there is better variant:

```
RewriteEngine on
RewriteCond  %{REQUEST_URI}  !-U
RewriteRule  ^(.+)           http://webserverB.dom/$1
```

This uses the URL look-ahead feature of `mod_rewrite`. The result is that this will work for all types of URLs and is a safe way. But it does a performance impact on the webserver, because for every request there is one more internal subrequest. So, if your webserver runs on a powerful CPU, use this one. If it is a slow machine, use the first approach or better a `ErrorDocument` CGI-script.

Extended Redirection

Description:

Sometimes we need more control (concerning the character escaping mechanism) of URLs on redirects. Usually the Apache kernels URL escape function also escapes anchors, i.e. URLs like `"url#anchor"`. You cannot use this directly on redirects with `mod_rewrite` because the `uri_escape()` function of Apache would also escape the hash character. How can we redirect to such a URL?

Solution:

We have to use a kludge by the use of a NPH-CGI script which does the redirect itself. Because here no escaping is done (NPH=non-parseable headers). First we introduce a new URL scheme `xredirect:` by the following per-server config-line (should be one of the last rewrite rules):

```
RewriteRule ^xredirect:(.+) /path/to/nph-xredirect.cgi/$1 \
[T=application/x-httpd-cgi,L]
```

This forces all URLs prefixed with `xredirect:` to be piped through the `nph-xredirect.cgi` program. And this program just looks like:

```
#!/path/to/perl
##
##  nph-xredirect.cgi -- NPH/CGI script for extended redirects
##  Copyright (c) 1997 Ralf S. Engelschall, All Rights Reserved.
##

$| = 1;
$url = $ENV{'PATH_INFO'};

print "HTTP/1.0 302 Moved Temporarily\n";
print "Server: $ENV{'SERVER_SOFTWARE'}\n";
print "Location: $url\n";
print "Content-type: text/html\n";
print "\n";
print "<html>\n";
print "<head>\n";
print "<title>302 Moved Temporarily (EXTENDED)</title>\n";
print "</head>\n";
print "<body>\n";
print "<h1>Moved Temporarily (EXTENDED)</h1>\n";
print "The document has moved <a HREF=\"$url\">here</a>.<p>\n";
print "</body>\n";
print "</html>\n";

##EOF##
```

This provides you with the functionality to do redirects to all URL schemes, i.e. including the one which are not directly accepted by mod_rewrite. For instance you can now also redirect to news: newsgroup via

```
RewriteRule ^anyurl xredirect:news:newsgroup
```

Notice: You have not to put [R] or [R,L] to the above rule because the xredirect: need to be expanded later by our special "pipe through" rule above.

Archive Access Multiplexer

Description:

Do you know the great CPAN (Comprehensive Perl Archive Network) under <http://www.perl.com/CPAN>? This does a redirect to one of several FTP servers around the world which carry a CPAN mirror and is approximately near the location of the requesting client. Actually this can be called an FTP access multiplexing service. While CPAN runs via CGI scripts, how can a similar approach implemented via mod_rewrite?

Solution:

First we notice that from version 3.0.0 mod_rewrite can also use the "ftp:" scheme on redirects. And second, the location approximation can be done by a rewritesmap over the top-level domain of the client. With a tricky chained ruleset we can use this top-level domain as a key to our multiplexing map.

```
RewriteEngine on
RewriteMap    multiplex          txt:/path/to/map.cxan
RewriteRule   ^/CxAN/(.*)       %{REMOTE_HOST}:::$1      [C]
RewriteRule   ^.+\.([a-zA-Z]+)::(.*)$  ${multiplex:$1|ftp.default.dom}$2  [R,L]
```

```
##
##  map.cxan -- Multiplexing Map for CxAN
##

de      ftp://ftp.cxan.de/CxAN/
uk      ftp://ftp.cxan.uk/CxAN/
com     ftp://ftp.cxan.com/CxAN/
:
##EOF##
```

Time-Dependent Rewriting

Description:

When tricks like time-dependent content should happen a lot of webmasters still use CGI scripts which do for instance redirects to specialized pages. How can it be done via mod_rewrite?

Solution:

There are a lot of variables named `TIME_xxx` for rewrite conditions. In conjunction with the special lexicographic comparison patterns `<STRING`, `>STRING` and `=STRING` we can do time-dependent redirects:

```
RewriteEngine on
RewriteCond  %{TIME_HOUR}%{TIME_MIN} >0700
RewriteCond  %{TIME_HOUR}%{TIME_MIN} <1900
RewriteRule  ^foo\.html$                foo.day.html
RewriteRule  ^foo\.html$                foo.night.html
```

This provides the content of `foo.day.html` under the URL `foo.html` from 07:00-19:00 and at the remaining time the contents of `foo.night.html`. Just a nice feature for a homepage...

Backward Compatibility for YYYY to XXXX migration

Description:

How can we make URLs backward compatible (still existing virtually) after migrating document.YYYY to document.XXXX, e.g. after translating a bunch of .html files to .phtml?

Solution:

We just rewrite the name to its basename and test for existence of the new extension. If it exists, we take that name, else we rewrite the URL to its original state.

```
# backward compatibility ruleset for
# rewriting document.html to document.phtml
# when and only when document.phtml exists
# but no longer document.html
RewriteEngine on
RewriteBase  /~quux/
# parse out basename, but remember the fact
RewriteRule  ^(.*)\.html$                $1          [C,E=WasHTML:yes]
# rewrite to document.phtml if exists
RewriteCond  %{REQUEST_FILENAME}.phtml -f
RewriteRule  ^(.*)$ $1.phtml              [S=1]
# else reverse the previous basename cutout
RewriteCond  %{ENV:WasHTML}              ^yes$
RewriteRule  ^(.*)$ $1.html
```

Content Handling

From Old to New (intern)

Description:

Assume we have recently renamed the page `bar.html` to `foo.html` and now want to provide the old URL for backward compatibility. Actually we want that users of the old URL even not recognize that the pages was renamed.

Solution:

We rewrite the old URL to the new one internally via the following rule:

```

RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo\.html$ bar.html

```

From Old to New (extern)

Description:

Assume again that we have recently renamed the page `bar.html` to `foo.html` and now want to provide the old URL for backward compatibility. But this time we want that the users of the old URL get hinted to the new one, i.e. their browsers Location field should change, too.

Solution:

We force a HTTP redirect to the new URL which leads to a change of the browsers and thus the users view:

```

RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo\.html$ bar.html [R]

```

Browser Dependend Content

Description:

At least for important top-level pages it is sometimes necessary to provide the optimum of browser dependend content, i.e. one has to provide a maximum version for the latest Netscape variants, a minimum version for the Lynx browsers and a average feature version for all others.

Solution:

We cannot use content negotiation because the browsers do not provide their type in that form. Instead we have to act on the HTTP header "User-Agent". The following condig does the following: If the HTTP header "User-Agent" begins with "Mozilla/3", the page `foo.html` is rewritten to `foo.NS.html` and and the rewriting stops. If the browser is "Lynx" or "Mozilla" of version 1 or 2 the URL becomes `foo.20.html`. All other browsers receive page `foo.32.html`. This is done by the following ruleset:

```

RewriteCond %{HTTP_USER_AGENT} ^Mozilla/3.*
RewriteRule ^foo\.html$ foo.NS.html [L]

RewriteCond %{HTTP_USER_AGENT} ^Lynx/. * [OR]
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/[12]. *
RewriteRule ^foo\.html$ foo.20.html [L]

RewriteRule ^foo\.html$ foo.32.html [L]

```

Dynamic Mirror

Description:

Assume there are nice webpages on remote hosts we want to bring into our namespace. For FTP servers we would use the `mirror` program which actually maintains an explicit up-to-date copy of the remote data on the local machine. For a webserver we could use the program `webcopy` which acts similar via HTTP. But both techniques have one major drawback: The local copy is always just as up-to-date as often we run the program. It would be much better if the mirror is not a static one we have to establish explicitly. Instead we want a dynamic mirror with data which gets updated automatically when there is need (updated data on the remote host).

Solution:

To provide this feature we map the remote webpage or even the complete remote webarea to our namespace by the use of the *Proxy Throughput* feature (flag [P]):

```

RewriteEngine on
RewriteBase /~quux/
RewriteRule ^hotsheet/(.*)$ http://www.tstimpreso.com/hotsheet/$1 [P]

```

```

RewriteEngine on
RewriteBase    /~quux/
RewriteRule    ^usa-news\.html$    http://www.quux-corp.com/news/index.html [P]

```

Reverse Dynamic Mirror

Description:

...

Solution:

```

RewriteEngine on
RewriteCond    /mirror/of/remotesite/$1    -U
RewriteRule    ^http://www\.remotesite\.com/(.*)$ /mirror/of/remotesite/$1

```

Retrieve Missing Data from Intranet

Description:

This is a tricky way of virtually running a corporates (external) Internet webserver (`www.quux-corp.dom`), while actually keeping and maintaining its data on a (internal) Intranet webserver (`www2.quux-corp.dom`) which is protected by a firewall. The trick is that on the external webserver we retrieve the requested data on-the-fly from the internal one.

Solution:

First, we have to make sure that our firewall still protects the internal webserver and that only the external webserver is allowed to retrieve data from it. For a packet-filtering firewall we could for instance configure a firewall ruleset like the following:

```

ALLOW Host www.quux-corp.dom Port >1024 --> Host www2.quux-corp.dom Port 80
DENY   Host *                      Port *          --> Host www2.quux-corp.dom Port 80

```

Just adjust it to your actual configuration syntax. Now we can establish the `mod_rewrite` rules which request the missing data in the background through the proxy throughput feature:

```

RewriteRule    ^/~([^/]+)/?(.*)    /home/$1/.www/$2
RewriteCond    %{REQUEST_FILENAME}    !-f
RewriteCond    %{REQUEST_FILENAME}    !-d
RewriteRule    ^/home/([^/]+)/.www/?(.*)    http://www2.quux-corp.dom/~$1/pub/$2 [P]

```

Load Balancing

Description:

Suppose we want to load balance the traffic to `www.foo.com` over `www[0-5].foo.com` (a total of 6 servers). How can this be done?

Solution:

There are a lot of possible solutions for this problem. We will discuss first a commonly known DNS-based variant and then the special one with `mod_rewrite`:

1. DNS Round-Robin

The simplest method for load-balancing is to use the DNS round-robin feature of BIND. Here you just configure `www[0-9].foo.com` as usual in your DNS with A(address) records, e.g.

```

www0    IN    A      1.2.3.1
www1    IN    A      1.2.3.2
www2    IN    A      1.2.3.3
www3    IN    A      1.2.3.4
www4    IN    A      1.2.3.5
www5    IN    A      1.2.3.6

```

Then you additionally add the following entry:

```

www      IN    CNAME  www0.foo.com.
         IN    CNAME  www1.foo.com.
         IN    CNAME  www2.foo.com.
         IN    CNAME  www3.foo.com.
         IN    CNAME  www4.foo.com.
         IN    CNAME  www5.foo.com.
         IN    CNAME  www6.foo.com.

```

Notice that this seems wrong, but is actually an intended feature of BIND and can be used in this way. However, now when `www.foo.com` gets resolved, BIND gives out `www0-www6` - but in a slightly permuted/rotated order every time. This way the clients are spread over the various servers. But notice that this not a perfect load balancing scheme, because DNS resolve information gets cached by the other nameservers on the net, so once a client has resolved `www.foo.com` to a particular `wwwN.foo.com`, all subsequent requests also go to this particular name `wwwN.foo.com`. But the final result is ok, because the total sum of the requests are really spread over the various web servers.

2. DNS Load-Balancing

A sophisticated DNS-based method for load-balancing is to use the program `lbname` which can be found at <http://www.stanford.edu/~schemers/docs/lbname/lbname.html>. It is a Perl 5 program in conjunction with auxiliary tools which provides a real load-balancing for DNS.

3. Proxy Throughput Round-Robin

In this variant we use `mod_rewrite` and its proxy throughput feature. First we dedicate `www0.foo.com` to be actually `www.foo.com` by using a single

```

www      IN    CNAME  www0.foo.com.

```

entry in the DNS. Then we convert `www0.foo.com` to a proxy-only server, i.e. we configure this machine so all arriving URLs are just pushed through the internal proxy to one of the 5 other servers (`www1-www5`). To accomplish this we first establish a ruleset which contacts a load balancing script `lb.pl` for all URLs.

```

RewriteEngine on
RewriteMap    lb      prg:/path/to/lb.pl
RewriteRule   ^/(.+)$ ${lb:$1}          [P,L]

```

Then we write `lb.pl`:

```
#!/path/to/perl
##
## lb.pl -- load balancing script
##

$| = 1;

$name = "www";      # the hostname base
$first = 1;         # the first server (not 0 here, because 0 is myself)
$last = 5;          # the last server in the round-robin
$domain = "foo.dom"; # the domainname

$cnt = 0;
while (<STDIN>) {
    $cnt = (($cnt+1) % ($last+1-$first));
    $server = sprintf("%s%d.%s", $name, $cnt+$first, $domain);
    print "http://$server/$_";
}

##EOF##
```

A last notice: Why is this useful? Seems like `www0.foo.com` still is overloaded? The answer is yes, it is overloaded, but with plain proxy throughput requests, only! All SSI, CGI, ePerl, etc. processing is completely done on the other machines. This is the essential point.

4. Hardware/TCP Round-Robin

There is a hardware solution available, too. Cisco has a beast called LocalDirector which does a load balancing at the TCP/IP level. Actually this is some sort of a circuit level gateway in front of a webcluster. If you have enough money and really need a solution with high performance, use this one.

Reverse Proxy

Description:

...

Solution:

```
##
## apache-rproxy.conf -- Apache configuration for Reverse Proxy Usage
##

# server type
ServerType standalone
Port 8000
MinSpareServers 16
StartServers 16
MaxSpareServers 16
MaxClients 16
MaxRequestsPerChild 100

# server operation parameters
KeepAlive on
MaxKeepAliveRequests 100
KeepAliveTimeout 15
Timeout 400
IdentityCheck off
HostnameLookups off

# paths to runtime files
PidFile /path/to/apache-rproxy.pid
LockFile /path/to/apache-rproxy.lock
ErrorLog /path/to/apache-rproxy.elog
CustomLog /path/to/apache-rproxy.dlog "%{%v/%T}t %h -> %{SERVER}e URL: %U"
```

```
# unused paths
ServerRoot      /tmp
DocumentRoot    /tmp
CacheRoot       /tmp
RewriteLog      /dev/null
TransferLog     /dev/null
TypesConfig     /dev/null
AccessConfig    /dev/null
ResourceConfig  /dev/null

# speed up and secure processing
<Directory />
Options -FollowSymLinks -SymLinksIfOwnerMatch
AllowOverride None
</Directory>

# the status page for monitoring the reverse proxy
<Location /rproxy-status>
SetHandler server-status
</Location>

# enable the URL rewriting engine
RewriteEngine    on
RewriteLogLevel  0

# define a rewriting map with value-lists where
# mod_rewrite randomly chooses a particular value
RewriteMap       server rnd:/path/to/apache-rproxy.conf-servers

# make sure the status page is handled locally
# and make sure no one uses our proxy except ourself
RewriteRule     ^/apache-rproxy-status.* - [L]
RewriteRule     ^(http|ftp)://.*        - [F]

# now choose the possible servers for particular URL types
RewriteRule     ^/(.*\.(cgi|shtml))$    to://${server:dynamic}/${1} [S=1]
RewriteRule     ^/(.*)$                 to://${server:static}/${1}

# and delegate the generated URL by passing it
# through the proxy module
RewriteRule     ^to://([^/]+)/(.*)      http://$1/$2 [E=SERVER:$1,P,L]

# and make really sure all other stuff is forbidden
# when it should survive the above rules...
RewriteRule     .*                       - [F]

# enable the Proxy module without caching
ProxyRequests   on
NoCache         *

# setup URL reverse mapping for redirect responses
ProxyPassReverse / http://www1.foo.dom/
ProxyPassReverse / http://www2.foo.dom/
ProxyPassReverse / http://www3.foo.dom/
ProxyPassReverse / http://www4.foo.dom/
ProxyPassReverse / http://www5.foo.dom/
ProxyPassReverse / http://www6.foo.dom/
```

```
##
##  apache-rproxy.conf-servers -- Apache/mod_rewrite selection table
##

#  list of backend servers which serve static
#  pages (HTML files and Images, etc.)
static    www1.foo.dom|www2.foo.dom|www3.foo.dom|www4.foo.dom

#  list of backend servers which serve dynamically
#  generated page (CGI programs or mod_perl scripts)
dynamic   www5.foo.dom|www6.foo.dom
```

New MIME-type, New Service

Description:

On the net there are a lot of nifty CGI programs. But their usage is usually boring, so a lot of webmaster don't use them. Even Apache's Action handler feature for MIME-types is only appropriate when the CGI programs don't need special URLs (actually `PATH_INFO` and `QUERY_STRING`s) as their input. First, let us configure a new file type with extension `.scgi` (for secure CGI) which will be processed by the popular `cgiwrap` program. The problem here is that for instance we use a Homogeneous URL Layout (see above) a file inside the user homedirs has the URL `/u/user/foo/bar.scgi`. But `cgiwrap` needs the URL in the form `/~user/foo/bar.scgi/`. The following rule solves the problem:

```
RewriteRule ^/[uge]/([^/]+)/\.www/(+)\.scgi(.*)...
... /internal/cgi/user/cgiwrap/~$1/$2.scgi$3 [NS,T=application/x-http-cgi]
```

Or assume we have some more nifty programs: `wwwlog` (which displays the `access.log` for a URL subtree and `wwidx` (which runs `Glimpse` on a URL subtree). We have to provide the URL area to these programs so they know on which area they have to act on. But usually this ugly, because they are all the times still requested from that areas, i.e. typically we would run the `swidx` program from within `/u/user/foo/` via hyperlink to

```
/internal/cgi/user/swidx?i=/u/user/foo/
```

which is ugly. Because we have to hard-code **both** the location of the area **and** the location of the CGI inside the hyperlink. When we have to reorganise or area, we spend a lot of time changing the various hyperlinks.

Solution:

The solution here is to provide a special new URL format which automatically leads to the proper CGI invocation. We configure the following:

```
RewriteRule ^/([uge])/([^/]+)(/?.*)/* /internal/cgi/user/wwidx?i=/$1/$2$3/
RewriteRule ^/([uge])/([^/]+)(/?.*):log /internal/cgi/user/wwwlog?f=/$1/$2$3
```

Now the hyperlink to search at `/u/user/foo/` reads only

```
HREF=" * "
```

which internally gets automatically transformed to

```
/internal/cgi/user/wwidx?i=/u/user/foo/
```

The same approach leads to an invocation for the access log CGI program when the hyperlink `:log` gets used.

From Static to Dynamic

Description:

How can we transform a static page `foo.html` into a dynamic variant `foo.cgi` in a seamless way, i.e. without notice by the browser/user.

Solution:

We just rewrite the URL to the CGI-script and force the correct MIME-type so it gets really run as a CGI-script. This way a request to `/~quux/foo.html` internally leads to the invocation of `/~quux/foo.cgi`.

```
RewriteEngine on
RewriteBase /~quux/
RewriteRule ^foo\.html$ foo.cgi [T=application/x-httpd-cgi]
```

On-the-fly Content-Regeneration

Description:

Here comes a really esoteric feature: Dynamically generated but statically served pages, i.e. pages should be delivered as pure static pages (read from the filesystem and just passed through), but they have to be generated dynamically by the webserver if missing. This way you can have CGI-generated pages which are statically served unless one (or a cronjob) removes the static contents. Then the contents gets refreshed.

Solution:

This is done via the following ruleset:

```
RewriteCond %{REQUEST_FILENAME} !-s
RewriteRule ^page\.html$ page.cgi [T=application/x-httpd-cgi,L]
```

Here a request to `page.html` leads to an internal run of a corresponding `page.cgi` if `page.html` is still missing or has filesize null. The trick here is that `page.cgi` is a usual CGI script which (additionally to its `STDOUT`) writes its output to the file `page.html`. Once it was run, the server sends out the data of `page.html`. When the webmaster wants to force a refresh the contents, he just removes `page.html` (usually done by a cronjob).

Document With Autorefresh

Description:

Wouldn't it be nice while creating a complex webpage if the webbrowser would automatically refresh the page every time we write a new version from within our editor? Impossible?

Solution:

No! We just combine the MIME multipart feature, the webserver NPH feature and the URL manipulation power of `mod_rewrite`. First, we establish a new URL feature: Adding just `:refresh` to any URL causes this to be refreshed every time it gets updated on the filesystem.

```
RewriteRule ^(/[uqe]/[^\s]+/?.*):refresh /internal/cgi/apache/nph-refresh?f=$1
```

Now when we reference the URL

```
/u/foo/bar/page.html:refresh
```

this leads to the internal invocation of the URL

```
/internal/cgi/apache/nph-refresh?f=/u/foo/bar/page.html
```

The only missing part is the NPH-CGI script. Although one would usually say "left as an exercise to the reader" ;-) I will provide this, too.

```
#!/sw/bin/perl
##
## nph-refresh -- NPH/CGI script for auto refreshing pages
## Copyright (c) 1997 Ralf S. Engelschall, All Rights Reserved.
##
$| = 1;

# split the QUERY_STRING variable
@pairs = split(/&/, $ENV{'QUERY_STRING'});
foreach $pair (@pairs) {
```

```

        ($name, $value) = split(/=/, $pair);
        $name =~ tr/A-Z/a-z/;
        $name = 'QS_' . $name;
        $value =~ s/%([a-fA-F0-9][a-fA-F0-9])/pack("C", hex($1))/eg;
        eval "\$$name = \"\$value\"";
    }
    $QS_s = 1 if ($QS_s eq '');
    $QS_n = 3600 if ($QS_n eq '');
    if ($QS_f eq '') {
        print "HTTP/1.0 200 OK\n";
        print "Content-type: text/html\n\n";
        print "&lt;b&gt;ERROR&lt;/b&gt;: No file given\n";
        exit(0);
    }
    if (! -f $QS_f) {
        print "HTTP/1.0 200 OK\n";
        print "Content-type: text/html\n\n";
        print "&lt;b&gt;ERROR&lt;/b&gt;: File $QS_f not found\n";
        exit(0);
    }

    sub print_http_headers_multipart_begin {
        print "HTTP/1.0 200 OK\n";
        $bound = "ThisRandomString12345";
        print "Content-type: multipart/x-mixed-replace;boundary=$bound\n";
        &print_http_headers_multipart_next;
    }

    sub print_http_headers_multipart_next {
        print "\n--$bound\n";
    }

    sub print_http_headers_multipart_end {
        print "\n--$bound--\n";
    }

    sub displayhtml {
        local($buffer) = @_;
        $len = length($buffer);
        print "Content-type: text/html\n";
        print "Content-length: $len\n\n";
        print $buffer;
    }

    sub readfile {
        local($file) = @_;
        local(*FP, $size, $buffer, $bytes);
        ($x, $x, $x, $x, $x, $x, $x, $size) = stat($file);
        $size = sprintf("%d", $size);
        open(FP, "&lt;$file");
        $bytes = sysread(FP, $buffer, $size);
        close(FP);
        return $buffer;
    }

    $buffer = &readfile($QS_f);
    &print_http_headers_multipart_begin;
    &displayhtml($buffer);

    sub mystat {
        local($file) = $_[0];
        local($time);

        ($x, $x, $x, $x, $x, $x, $x, $x, $x, $x, $mtime) = stat($file);
        return $mtime;
    }

    $mtimeL = &mystat($QS_f);

```

```

$mtime = $mtime;
for ($n = 0; $n < $QS_n; $n++) {
    while (1) {
        $mtime = &mystat($QS_f);
        if ($mtime ne $mtimeL) {
            $mtimeL = $mtime;
            sleep(2);
            $buffer = &readfile($QS_f);
            &print_http_headers_multipart_next;
            &displayhtml($buffer);
            sleep(5);
            $mtimeL = &mystat($QS_f);
            last;
        }
        sleep($QS_s);
    }
}

&print_http_headers_multipart_end;

exit(0);

##EOF##

```

Mass Virtual Hosting

Description:

The <VirtualHost> feature of Apache is nice and works great when you just have a few dozens virtual hosts. But when you are an ISP and have hundreds of virtual hosts to provide this feature is not the best choice.

Solution:

To provide this feature we map the remote webpage or even the complete remote webarea to our namespace by the use of the *Proxy Throughput* feature (flag [P]):

```

##
## vhost.map
##
www.vhost1.dom:80 /path/to/docroot/vhost1
www.vhost2.dom:80 /path/to/docroot/vhost2
:
www.vhostN.dom:80 /path/to/docroot/vhostN

```

```

##
## httpd.conf
##
:
# use the canonical hostname on redirects, etc.
UseCanonicalName on

:
# add the virtual host in front of the CLF-format
CustomLog /path/to/access_log "%{VHOST}e %h %l %u %t \"%r\" %>s %b"
:

# enable the rewriting engine in the main server
RewriteEngine on

# define two maps: one for fixing the URL and one which defines
# the available virtual hosts with their corresponding
# DocumentRoot.
RewriteMap lowercase int:tolower
RewriteMap vhost txt:/path/to/vhost.map

# Now do the actual virtual host mapping

```

```

#   via a huge and complicated single rule:
#
#   1. make sure we don't map for common locations
RewriteCond  %{REQUEST_URI}  !^/commonurl1/*
RewriteCond  %{REQUEST_URI}  !^/commonurl2/*
      :
RewriteCond  %{REQUEST_URI}  !^/commonurlN/*
#
#   2. make sure we have a Host header, because
#       currently our approach only supports
#       virtual hosting through this header
RewriteCond  %{HTTP_HOST}    !^$
#
#   3. lowercase the hostname
RewriteCond  ${lowercase:%{HTTP_HOST}|NONE}  ^(.+)$
#
#   4. lookup this hostname in vhost.map and
#       remember it only when it is a path
#       (and not "NONE" from above)
RewriteCond  ${vhost:%1}     ^(/.*)$
#
#   5. finally we can map the URL to its docroot location
#       and remember the virtual host for logging puposes
RewriteRule  ^/(.*)$         %1/$1  [E=VHOST:${lowercase:%{HTTP_HOST}}]
      :

```

Access Restriction

Blocking of Robots

Description:

How can we block a really annoying robot from retrieving pages of a specific webarea? A `/robots.txt` file containing entries of the "Robot Exclusion Protocol" is typically not enough to get rid of such a robot.

Solution:

We use a ruleset which forbids the URLs of the webarea `/~quux/foo/arc/` (perhaps a very deep directory indexed area where the robot traversal would create big server load). We have to make sure that we forbid access only to the particular robot, i.e. just forbidding the host where the robot runs is not enough. This would block users from this host, too. We accomplish this by also matching the User-Agent HTTP header information.

```

RewriteCond  %{HTTP_USER_AGENT}    ^NameOfBadRobot.*
RewriteCond  %{REMOTE_ADDR}        ^123\.45\.67\.[8-9]$
RewriteRule  ^/~quux/foo/arc/.+    -    [F]

```

Blocked Inline-Images

Description:

Assume we have under `http://www.quux-corp.de/~quux/` some pages with inlined GIF graphics. These graphics are nice, so others directly incorporate them via hyperlinks to their pages. We don't like this practice because it adds useless traffic to our server.

Solution:

While we cannot 100% protect the images from inclusion, we can at least restrict the cases where the browser sends a HTTP Referer header.

```

RewriteCond  %{HTTP_REFERER}    !^$
RewriteCond  %{HTTP_REFERER}    !^http://www.quux-corp.de/~quux/.*$ [NC]
RewriteRule  *.*\.gif$          -    [F]

```

```

RewriteCond %{HTTP_REFERER}          !^$
RewriteCond %{HTTP_REFERER}          !.*\/foo-with-gif\.html$
RewriteRule ^inlined-in-foo\.gif$    - [F]

```

Host Deny

Description:

How can we forbid a list of externally configured hosts from using our server?

Solution:

For Apache >= 1.3b6:

```

RewriteEngine on
RewriteMap     hosts-deny  txt:/path/to/hosts.deny
RewriteCond    ${hosts-deny:%{REMOTE_HOST}|NOT-FOUND} !=NOT-FOUND [OR]
RewriteCond    ${hosts-deny:%{REMOTE_ADDR}|NOT-FOUND} !=NOT-FOUND
RewriteRule    ^/.* - [F]

```

For Apache <= 1.3b6:

```

RewriteEngine on
RewriteMap     hosts-deny  txt:/path/to/hosts.deny
RewriteRule    ^/(.*)$ ${hosts-deny:%{REMOTE_HOST}|NOT-FOUND}/$1
RewriteRule    !^NOT-FOUND/.* - [F]
RewriteRule    ^NOT-FOUND/(.*)$ ${hosts-deny:%{REMOTE_ADDR}|NOT-FOUND}/$1
RewriteRule    !^NOT-FOUND/.* - [F]
RewriteRule    ^NOT-FOUND/(.*)$ /$1

```

```

##
##  hosts.deny
##
##  ATTENTION! This is a map, not a list, even when we treat it as such.
##              mod_rewrite parses it for key/value pairs, so at least a
##              dummy value "-" must be present for each entry.
##
193.102.180.41 -
bsdtil.sdm.de -
192.76.162.40 -

```

URL-Restricted Proxy

Description:

How can we restrict the proxy to allow access to a configurable set of internet sites only? The site list is extracted from a prepared bookmarks file.

Solution:

We first have to make sure `mod_rewrite` is below(!) `mod_proxy` in the `Configuration` file when compiling the Apache webserver (or in the `AddModule` list of `httpd.conf` in the case of dynamically loaded modules), as it must get called `_before_` `mod_proxy`.

For simplicity, we generate the site list as a textfile map (but see the [mod_rewrite documentation](#) for a conversion script to DBM format). A typical Netscape bookmarks file can be converted to a list of sites with a shell script like this:

```
#!/bin/sh
cat ${1:~/\.netscape/bookmarks.html} |
tr -d '\015' | tr '[A-Z]' '[a-z]' | grep href=" |
sed -e '/href="file:/d;' -e '/href="news:/d;' \
    -e 's|^.*href="[^:]*://\([^:/*]*\).*$|\1 OK|;' \
    -e '/href="s|^.*href="\([^:/*]*\).*$|\1 OK|;' |
sort -u
```

We redirect the resulting output into a text file called `goodsites.txt`. It now looks similar to this:

```
www.apache.org OK
xml.apache.org OK
jakarta.apache.org OK
perl.apache.org OK
...
```

We reference this site file within the configuration for the `VirtualHost` which is responsible for serving as a proxy (often not port 80, but 81, 8080 or 8008).

```
<VirtualHost *:8008>
...
RewriteEngine On
# Either use the (plaintext) allow list from goodsites.txt
RewriteMap ProxyAllow txt:/usr/local/apache/conf/goodsites.txt
# Or, for faster access, convert it to a DBM database:
#RewriteMap ProxyAllow dbm:/usr/local/apache/conf/goodsites
# Match lowercased hostnames
RewriteMap lowercase int:tolower
# Here we go:
# 1) first lowercase the site name and strip off a :port suffix
RewriteCond ${lowercase:%{HTTP_HOST}} ^([^:]*).*$
# 2) next look it up in the map file.
# "%1" refers to the previous regex.
# If the result is "OK", proxy access is granted.
RewriteCond ${ProxyAllow:%1|DENY} !^OK$ [NC]
# 3) Disallow proxy requests if the site was not tagged "OK":
RewriteRule ^proxy: - [F]
...
</VirtualHost>
```

Proxy Deny

Description:

How can we forbid a certain host or even a user of a special host from using the Apache proxy?

Solution:

We first have to make sure `mod_rewrite` is below(!) `mod_proxy` in the `Configuration` file when compiling the Apache webserver. This way it gets called `_before_` `mod_proxy`. Then we configure the following for a host-dependent deny...

```
RewriteCond %{REMOTE_HOST} ^badhost\.mydomain\.com$
RewriteRule !^http://[^\./]\.mydomain.com.* - [F]
```

...and this one for a user@host-dependent deny:

```
RewriteCond %{REMOTE_IDENT}@%{REMOTE_HOST} ^badguy@badhost\.mydomain\.com$
RewriteRule !^http://[^\./]\.mydomain.com.* - [F]
```

Special Authentication Variant

Description:

Sometimes a very special authentication is needed, for instance a authentication which checks for a set of explicitly configured users. Only these should receive access and without explicit prompting (which would occur when using the Basic Auth via mod_access).

Solution:

We use a list of rewrite conditions to exclude all except our friends:

```
RewriteCond %{REMOTE_IDENT}@%{REMOTE_HOST} !^friend1@client1.quux-corp\.com$
RewriteCond %{REMOTE_IDENT}@%{REMOTE_HOST} !^friend2@client2.quux-corp\.com$
RewriteCond %{REMOTE_IDENT}@%{REMOTE_HOST} !^friend3@client3.quux-corp\.com$
RewriteRule ^/~quux/only-for-friends/ - [F]
```

Referer-based Deflector

Description:

How can we program a flexible URL Deflector which acts on the "Referer" HTTP header and can be configured with as many referring pages as we like?

Solution:

Use the following really tricky ruleset...

```
RewriteMap deflector txt:/path/to/deflector.map

RewriteCond %{HTTP_REFERER} !=" "
RewriteCond ${deflector:%{HTTP_REFERER}} ^-$
RewriteRule ^.* %{HTTP_REFERER} [R,L]

RewriteCond %{HTTP_REFERER} !=" "
RewriteCond ${deflector:%{HTTP_REFERER}|NOT-FOUND} !=NOT-FOUND
RewriteRule ^.* ${deflector:%{HTTP_REFERER}} [R,L]
```

... in conjunction with a corresponding rewrite map:

```
##
## deflector.map
##

http://www.badguys.com/bad/index.html -
http://www.badguys.com/bad/index2.html -
http://www.badguys.com/bad/index3.html http://somewhere.com/
```

This automatically redirects the request back to the referring page (when "-" is used as the value in the map) or to a specific URL (when an URL is specified in the map as the second argument).

Other

External Rewriting Engine

Description:

A FAQ: How can we solve the FOO/BAR/QUUX/etc. problem? There seems no solution by the use of mod_rewrite...

Solution:

Use an external rewrite map, i.e. a program which acts like a rewrite map. It is run once on startup of Apache receives the requested URLs on STDIN and has to put the resulting (usually rewritten) URL on STDOUT (same order!).

```
RewriteEngine on
RewriteMap    quux-map      prog:/path/to/map.quux.pl
RewriteRule   ^/~quux/(.*)$ /~quux/${quux-map:$1}
```

```
#!/path/to/perl

#  disable buffered I/O which would lead
#  to deadloops for the Apache server
$| = 1;

#  read URLs one per line from stdin and
#  generate substitution URL on stdout
while (<>) {
    s|^foo/|bar/|;
    print $_;
}
```

This is a demonstration-only example and just rewrites all URLs `/~quux/foo/...` to `/~quux/bar/...`. Actually you can program whatever you like. But notice that while such maps can be **used** also by an average user, only the system administrator can **define** it.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache Virtual Host documentation

The term Virtual Host refers to the practice of maintaining more than one server on one machine, as differentiated by their apparent hostname. For example, it is often desirable for companies sharing a web server to have their own domains, with web servers accessible as `www.company1.com` and `www.company2.com`, without requiring the user to know any extra path information.

Apache was one of the first servers to support IP-based virtual hosts right out of the box. Versions 1.1 and later of Apache support both, IP-based and name-based virtual hosts (vhosts). The latter variant of virtual hosts is sometimes also called host-based or non-IP virtual hosts.

Below is a list of documentation pages which explain all details of virtual host support in Apache version 1.3 and later.

Virtual Host Support

- [Name-based Virtual Hosts](#)
- [IP-based Virtual Hosts](#)
- [Virtual Host examples for common setups](#)
- [In-Depth Discussion of Virtual Host Matching](#)
- [File Descriptor Limits](#)
- [Dynamically Configured Mass Virtual Hosting](#)

Configuration directives

- [<VirtualHost>](#)
- [NameVirtualHost](#)
- [ServerName](#)
- [ServerAlias](#)
- [ServerPath](#)

Folks trying to debug their virtual host configuration may find the Apache `-S` command line switch useful. It will dump out a description of how Apache parsed the configuration file. Careful examination of the IP addresses and server names may help uncover configuration mistakes.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Name-based Virtual Host Support

This document describes when and how to use name-based virtual hosts.

- [Name-based vs. IP-based Virtual Hosts](#)
- [Using Name-based Virtual Hosts](#)
- [Compatibility With Older Browsers](#)

See also: [Virtual Host examples for common setups](#), [IP-based Virtual Host Support](#), [An In-Depth Discussion of Virtual Host Matching](#), and [Dynamically configured mass virtual hosting](#).

Name-based vs. IP-based Virtual Hosts

IP-based virtual hosts use the IP address of the connection to determine the correct virtual host to serve. Therefore you need to have a separate IP address for each host. With name-based virtual hosting, the server relies on the client to report the hostname as part of the HTTP headers. Using this technique, many different hosts can share the same IP address.

Name-based virtual hosting is usually simpler, since you need only configure your DNS server to map each hostname to the correct IP address and then configure the Apache HTTP Server to recognize the different hostnames. Name-based virtual hosting also eases the demand for scarce IP addresses. Therefore you should use name-based virtual hosting unless there is a specific reason to choose IP-based virtual hosting. Some reasons why you might consider using IP-based virtual hosting:

- Some ancient clients are not compatible with name-based virtual hosting. For name-based virtual hosting to work, the client must send the HTTP Host header. This is required by HTTP/1.1, and is implemented by all modern HTTP/1.0 browsers as an extension. If you need to support obsolete clients and still use name-based virtual hosting, a possible technique is discussed at the end of this document.
- Name-based virtual hosting cannot be used with SSL secure servers because of the nature of the SSL protocol.
- Some operating systems and network equipment implement bandwidth management techniques that cannot differentiate between hosts unless they are on separate IP addresses.

Using Name-based Virtual Hosts

Related Directives

[DocumentRoot](#)
[NameVirtualHost](#)
[ServerAlias](#)
[ServerName](#)
[ServerPath](#)
[VirtualHost](#)

To use name-based virtual hosting, you must designate the IP address (and possibly port) on the server that will be accepting requests for the hosts. This is configured using the [NameVirtualHost](#) directive. In the normal case where any and all IP addresses on the server should be used, you can use * as the argument to NameVirtualHost. (NameVirtualHost * will work only in version 1.3.13 and later.) Note that mentioning an IP address in a NameVirtualHost directive does not automatically make the

server listen to that IP address. See [Setting which addresses and ports Apache uses](#) for more details. In addition, any IP address specified here must be associated with a network interface on the server.

The next step is to create a [<VirtualHost>](#) block for each different host that you would like to serve. The argument to the `<VirtualHost>` directive should be the same as the argument to the `NameVirtualHost` directive (ie, an IP address, or `*` for all addresses). Inside each `<VirtualHost>` block, you will need at minimum a [ServerName](#) directive to designate which host is served and a [DocumentRoot](#) directive to show where in the filesystem the content for that host lives.

For example, suppose that both `www.domain.tld` and `www.otherdomain.tld` point at an IP address that the server is listening to. Then you simply add the following to `httpd.conf`:

```
NameVirtualHost *

<VirtualHost *>
ServerName www.domain.tld
DocumentRoot /www/domain
</VirtualHost>

<VirtualHost *>
ServerName www.otherdomain.tld
DocumentRoot /www/otherdomain
</VirtualHost>
```

You can alternatively specify an explicit IP address in place of the `*` in both the `NameVirtualHost` and `<VirtualHost>` directives. The IP address is required in version 1.3.12 and earlier.

Many servers want to be accessible by more than one name. This is possible with the [ServerAlias](#) directive, placed inside the `<VirtualHost>` section. For example if you add this to the first `<VirtualHost>` block above

```
ServerAlias domain.tld *.domain.tld
```

then requests for all hosts in the `domain.tld` domain will be served by the `www.domain.tld` virtual host. The wildcard characters `*` and `?` can be used to match names. Of course, you can't just make up names and place them in `ServerName` or `ServerAlias`. You must first have your DNS server properly configured to map those names to an IP address associated with your server.

Finally, you can fine-tune the configuration of the virtual hosts by placing other directives inside the `<VirtualHost>` containers. Most directives can be placed in these containers and will then change the configuration only of the relevant virtual host. To find out if a particular directive is allowed, check the [Context](#) of the directive. Configuration directives set in the *main server context* (outside any `<VirtualHost>` container) will be used only if they are not overridden by the virtual host settings.

Now when a request arrives, the server will first check if it is using an IP address that matches the `NameVirtualHost`. If it is, then it will look at each `<VirtualHost>` section with a matching IP address and try to find one where the `ServerName` or `ServerAlias` matches the requested hostname. If it finds one, then it uses the configuration for that server. If no matching virtual host is found, then **the first listed virtual host** that matches the IP address will be used.

As a consequence, the first listed virtual host is the *default* virtual host. The `DocumentRoot` from the *main server* will **never** be used when an IP address matches the `NameVirtualHost` directive. If you would like to have a special configuration for requests that do not match any particular virtual host, simply put that configuration in a `<VirtualHost>` container and list it first in the configuration file.

Compatibility with Older Browsers

As mentioned earlier, there are some clients who do not send the required data for the name-based virtual hosts to work properly. These clients will always be sent the pages from the first virtual host listed for that IP address (the primary name-based virtual host).

There is a possible workaround with the [ServerPath](#) directive, albeit a slightly cumbersome one:

Example configuration:

Name-based Virtual Hosts

```
NameVirtualHost 111.22.33.44
```

```
<VirtualHost 111.22.33.44>  
ServerName www.domain.tld  
ServerPath /domain  
DocumentRoot /web/domain  
</VirtualHost>
```

What does this mean? It means that a request for any URI beginning with `"/domain"` will be served from the virtual host `www.domain.tld`. This means that the pages can be accessed as `http://www.domain.tld/domain/` for all clients, although clients sending a `Host:` header can also access it as `http://www.domain.tld/`.

In order to make this work, put a link on your primary virtual host's page to `http://www.domain.tld/domain/`. Then, in the virtual host's pages, be sure to use either purely relative links (e.g., `"file.html"` or `"../icons/image.gif"`) or links containing the prefacing `/domain/` (e.g., `"http://www.domain.tld/domain/misc/file.html"` or `"/domain/misc/file.html"`).

This requires a bit of discipline, but adherence to these guidelines will, for the most part, ensure that your pages will work with all browsers, new and old.

See also: [ServerPath configuration example](#)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache IP-based Virtual Host Support

See also: [Name-based Virtual Hosts Support](#)

System requirements

As the term IP-based indicates, the server **must have a different IP address for each IP-based virtual host**. This can be achieved by the machine having several physical network connections, or by use of virtual interfaces which are supported by most modern operating systems (see system documentation for details, these are frequently called "ip aliases", and the "ifconfig" command is most commonly used to set them up).

How to set up Apache

There are two ways of configuring apache to support multiple hosts. Either by running a separate httpd daemon for each hostname, or by running a single daemon which supports all the virtual hosts.

Use multiple daemons when:

- There are security partitioning issues, such as company1 does not want anyone at company2 to be able to read their data except via the web. In this case you would need two daemons, each running with different [User](#), [Group](#), [Listen](#), and [ServerRoot](#) settings.
- You can afford the memory and [file descriptor requirements](#) of listening to every IP alias on the machine. It's only possible to [Listen](#) to the "wildcard" address, or to specific addresses. So if you have a need to listen to a specific address for whatever reason, then you will need to listen to all specific addresses. (Although one httpd could listen to N-1 of the addresses, and another could listen to the remaining address.)

Use a single daemon when:

- Sharing of the httpd configuration between virtual hosts is acceptable.
- The machine services a large number of requests, and so the performance loss in running separate daemons may be significant.

Setting up multiple daemons

Create a separate httpd installation for each virtual host. For each installation, use the [Listen](#) directive in the configuration file to select which IP address (or virtual host) that daemon services. e.g.

```
Listen www.smallco.com:80
```

It is recommended that you use an IP address instead of a hostname (see [DNS caveats](#)).

Setting up a single daemon with virtual hosts

For this case, a single httpd will service requests for the main server and all the virtual hosts. The [VirtualHost](#) directive in the configuration file is used to set the values of [ServerAdmin](#), [ServerName](#), [DocumentRoot](#), [ErrorLog](#) and [TransferLog](#) or [CustomLog](#) configuration directives to different values for each virtual host. e.g.

```
<VirtualHost www.smallco.com>
ServerAdmin webmaster@mail.smallco.com
DocumentRoot /groups/smallco/www
ServerName www.smallco.com
ErrorLog /groups/smallco/logs/error_log
TransferLog /groups/smallco/logs/access_log
</VirtualHost>

<VirtualHost www.baygroup.org>
ServerAdmin webmaster@mail.baygroup.org
DocumentRoot /groups/baygroup/www
ServerName www.baygroup.org
ErrorLog /groups/baygroup/logs/error_log
TransferLog /groups/baygroup/logs/access_log
</VirtualHost>
```

It is recommended that you use an IP address instead of a hostname (see [DNS caveats](#)).

Almost **any** configuration directive can be put in the VirtualHost directive, with the exception of directives that control process creation and a few other directives. To find out if a directive can be used in the VirtualHost directive, check the [Context](#) using the [directive index](#).

[User](#) and [Group](#) may be used inside a VirtualHost directive if the [suEXEC wrapper](#) is used.

SECURITY: When specifying where to write log files, be aware of some security risks which are present if anyone other than the user that starts Apache has write access to the directory where they are written. See the [security tips](#) document for details.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Dynamically configured mass virtual hosting

This document describes how to efficiently serve an arbitrary number of virtual hosts with Apache 1.3.

Contents:

- [Motivation](#)
- [Overview](#)
- [Simple dynamic virtual hosts](#)
- [A virtually hosted homepages system](#)
- [Using more than one virtual hosting system on the same server](#)
- [More efficient IP-based virtual hosting](#)
- [Using older versions of Apache](#)
- [Simple dynamic virtual hosts using mod_rewrite](#)
- [A homepages system using mod_rewrite](#)
- [Using a separate virtual host configuration file](#)

Motivation

The techniques described here are of interest if your `httpd.conf` contains many `<VirtualHost>` sections that are substantially the same, for example:

```
NameVirtualHost 111.22.33.44
<VirtualHost 111.22.33.44>
    ServerName                www.customer-1.com
    DocumentRoot              /www/hosts/www.customer-1.com/docs
    ScriptAlias /cgi-bin/     /www/hosts/www.customer-1.com/cgi-bin
</VirtualHost>
<VirtualHost 111.22.33.44>
    ServerName                www.customer-2.com
    DocumentRoot              /www/hosts/www.customer-2.com/docs
    ScriptAlias /cgi-bin/     /www/hosts/www.customer-2.com/cgi-bin
</VirtualHost>
# blah blah blah
<VirtualHost 111.22.33.44>
    ServerName                www.customer-N.com
    DocumentRoot              /www/hosts/www.customer-N.com/docs
    ScriptAlias /cgi-bin/     /www/hosts/www.customer-N.com/cgi-bin
</VirtualHost>
```

The basic idea is to replace all of the static `<VirtualHost>` configuration with a mechanism that works it out dynamically. This has a number of advantages:

1. Your configuration file is smaller so Apache starts faster and uses less memory.
2. Adding virtual hosts is simply a matter of creating the appropriate directories in the filesystem and entries in the DNS - you don't need to reconfigure or restart Apache.

The main disadvantage is that you cannot have a different log file for each virtual host; however if you have very many virtual hosts then doing this is dubious anyway because it eats file descriptors. It is better to log to a pipe or a fifo and arrange for the process at the other end to distribute the logs to the customers (it can also accumulate statistics, etc.).

Overview

A virtual host is defined by two pieces of information: its IP address, and the contents of the `Host :` header in the HTTP request. The dynamic mass virtual hosting technique is based on automatically inserting this information into the pathname of the file that is used to satisfy the request. This is done most easily using [mod_vhost_alias](#), but if you are using a version of Apache up to 1.3.6 then you must use [mod_rewrite](#). Both of these modules are disabled by default; you must enable one of them when configuring and building Apache if you want to use this technique.

A couple of things need to be 'faked' to make the dynamic virtual host look like a normal one. The most important is the server name which is used by Apache to generate self-referential URLs, etc. It is configured with the `ServerName` directive, and it is available to CGIs via the `SERVER_NAME` environment variable. The actual value used at run time is controlled by the [UseCanonicalName](#) setting. With `UseCanonicalName Off` the server name comes from the contents of the `Host :` header in the request. With `UseCanonicalName DNS` it comes from a reverse DNS lookup of the virtual host's IP address. The former setting is used for name-based dynamic virtual hosting, and the latter is used for IP-based hosting. If Apache cannot work out the server name because there is no `Host :` header or the DNS lookup fails then the value configured with `ServerName` is used instead.

The other thing to 'fake' is the document root (configured with `DocumentRoot` and available to CGIs via the `DOCUMENT_ROOT` environment variable). In a normal configuration this setting is used by the core module when mapping URIs to filenames, but when the server is configured to do dynamic virtual hosting that job is taken over by another module (either `mod_vhost_alias` or `mod_rewrite`) which has a different way of doing the mapping. Neither of these modules is responsible for setting the `DOCUMENT_ROOT` environment variable so if any CGIs or SSI documents make use of it they will get a misleading value.

Simple dynamic virtual hosts

This extract from `httpd.conf` implements the virtual host arrangement outlined in the [Motivation](#) section above, but in a generic fashion using `mod_vhost_alias`.

```
# get the server name from the Host: header
UseCanonicalName Off

# this log format can be split per-virtual-host based on the first field
LogFormat "%V %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

# include the server name in the filenames used to satisfy requests
VirtualDocumentRoot /www/hosts/%0/docs
VirtualScriptAlias /www/hosts/%0/cgi-bin
```

This configuration can be changed into an IP-based virtual hosting solution by just turning `UseCanonicalName Off` into `UseCanonicalName DNS`. The server name that is inserted into the filename is then derived from the IP address of the virtual host.

A virtually hosted homepages system

This is an adjustment of the above system tailored for an ISP's homepages server. Using a slightly more complicated configuration we can select substrings of the server name to use in the filename so that e.g. the documents for `www.user.isp.com` are found in `/home/user/`. It uses a single `cgi-bin` directory instead of one per virtual host.

```
# all the preliminary stuff is the same as above, then

# include part of the server name in the filenames
VirtualDocumentRoot /www/hosts/%2/docs

# single cgi-bin directory
ScriptAlias /cgi-bin/ /www/std-cgi/
```

There are examples of more complicated `VirtualDocumentRoot` settings in [the `mod_vhost_alias` documentation](#).

Using more than one virtual hosting system on the same server

With more complicated setups you can use Apache's normal `<VirtualHost>` directives to control the scope of the various virtual hosting configurations. For example, you could have one IP address for homepages customers and another for commercial customers with the following setup. This can of course be combined with conventional `<VirtualHost>` configuration sections.

```
UseCanonicalName Off

LogFormat "%V %h %l %u %t \"%r\" %s %b" vcommon

<Directory /www/commercial>
    Options FollowSymLinks
    AllowOverride All
</Directory>

<Directory /www/homepages>
    Options FollowSymLinks
    AllowOverride None
</Directory>

<VirtualHost 111.22.33.44>
    ServerName www.commercial.isp.com

    CustomLog logs/access_log.commercial vcommon

    VirtualDocumentRoot /www/commercial/%0/docs
    VirtualScriptAlias /www/commercial/%0/cgi-bin
</VirtualHost>

<VirtualHost 111.22.33.45>
    ServerName www.homepages.isp.com

    CustomLog logs/access_log.homepages vcommon

    VirtualDocumentRoot /www/homepages/%0/docs
    ScriptAlias /cgi-bin/ /www/std-cgi/
</VirtualHost>
```

More efficient IP-based virtual hosting

After [the first example](#) I noted that it is easy to turn it into an IP-based virtual hosting setup. Unfortunately that configuration is not very efficient because it requires a DNS lookup for every request. This can be avoided by laying out the filesystem according to the IP addresses themselves rather than the corresponding names and changing the logging similarly. Apache will then usually not need to work out the server name and so incur a DNS lookup.

```
# get the server name from the reverse DNS of the IP address
UseCanonicalName DNS

# include the IP address in the logs so they may be split
LogFormat "%A %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

# include the IP address in the filenames
VirtualDocumentRootIP /www/hosts/%0/docs
VirtualScriptAliasIP /www/hosts/%0/cgi-bin
```

Using older versions of Apache

The examples above rely on `mod_vhost_alias` which appeared after version 1.3.6. If you are using a version of Apache without `mod_vhost_alias` then you can implement this technique with `mod_rewrite` as illustrated below, but only for Host:-header-based virtual hosts.

In addition there are some things to beware of with logging. Apache 1.3.6 is the first version to include the `%V` log format directive; in versions 1.3.0 - 1.3.3 the `%v` option did what `%V` does; version 1.3.4 has no equivalent. In all these versions of Apache the `UseCanonicalName` directive can appear in `.htaccess` files which means that customers can cause the wrong thing to be logged. Therefore the best thing to do is use the `%{Host}i` directive which logs the `Host:` header directly; note that this may include `:port` on the end which is not the case for `%V`.

Simple dynamic virtual hosts using `mod_rewrite`

This extract from `httpd.conf` does the same thing as [the first example](#). The first half is very similar to the corresponding part above but with some changes for backward compatibility and to make the `mod_rewrite` part work properly; the second half configures `mod_rewrite` to do the actual work.

There are a couple of especially tricky bits: By default, `mod_rewrite` runs before the other URI translation modules (`mod_alias` etc.) so if they are used then `mod_rewrite` must be configured to accommodate them. Also, some magic must be performed to do a per-dynamic-virtual-host equivalent of `ScriptAlias`.

```
# get the server name from the Host: header
UseCanonicalName Off

# splittable logs
LogFormat "%{Host}i %h %l %u %t \"%r\" %s %b" vcommon
CustomLog logs/access_log vcommon

<Directory /www/hosts>
    # ExecCGI is needed here because we can't force
    # CGI execution in the way that ScriptAlias does
    Options FollowSymLinks ExecCGI
</Directory>

# now for the hard bit
```

```
RewriteEngine On

# a ServerName derived from a Host: header may be any case at all
RewriteMap lowercase int:tolower

## deal with normal documents first:
# allow Alias /icons/ to work - repeat for other aliases
RewriteCond %{REQUEST_URI} !^/icons/
# allow CGIs to work
RewriteCond %{REQUEST_URI} !^/cgi-bin/
# do the magic
RewriteRule ^/(.*)$ /www/hosts/${lowercase:%{SERVER_NAME}}/docs/$1

## and now deal with CGIs - we have to force a MIME type
RewriteCond %{REQUEST_URI} ^/cgi-bin/
RewriteRule ^/(.*)$ /www/hosts/${lowercase:%{SERVER_NAME}}/cgi-bin/$1
[T=application/x-httpd-cgi]

# that's it!
```

A homepages system using mod_rewrite

This does the same thing as [the second example](#).

```
RewriteEngine on

RewriteMap lowercase int:tolower

# allow CGIs to work
RewriteCond %{REQUEST_URI} !^/cgi-bin/

# check the hostname is right so that the RewriteRule works
RewriteCond ${lowercase:%{SERVER_NAME}} ^www\.[a-z-]+\\.isp\.com$

# concatenate the virtual host name onto the start of the URI
# the [C] means do the next rewrite on the result of this one
RewriteRule ^(.+) ${lowercase:%{SERVER_NAME}}$1 [C]

# now create the real file name
RewriteRule ^www\.[a-z-]+\\.isp\.com/(.*) /home/$1/$2

# define the global CGI directory
ScriptAlias /cgi-bin/ /www/std-cgi/
```

Using a separate virtual host configuration file

This arrangement uses more advanced mod_rewrite features to get the translation from virtual host to document root from a separate configuration file. This provides more flexibility but requires more complicated configuration.

The vhost.map file contains something like this:

```
www.customer-1.com /www/customers/1
www.customer-2.com /www/customers/2
# ...
www.customer-N.com /www/customers/N
```

The http.conf contains this:

```
RewriteEngine on

RewriteMap lowercase int:tolower

# define the map file
RewriteMap vhost txt:/www/conf/vhost.map

# deal with aliases as above
RewriteCond %{REQUEST_URI} !^/icons/
RewriteCond %{REQUEST_URI} !^/cgi-bin/
RewriteCond ${lowercase:%{SERVER_NAME}} ^(.+)$
# this does the file-based remap
RewriteCond ${vhost:%1} ^(/.*)$
RewriteRule ^(/.*)$ %1/docs/$1

RewriteCond %{REQUEST_URI} ^/cgi-bin/
RewriteCond ${lowercase:%{SERVER_NAME}} ^(.+)$
RewriteCond ${vhost:%1} ^(/.*)$
RewriteRule ^(/.*)$ %1/cgi-bin/$1
```

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Virtual Host examples for common setups

Base configuration

- [Simple name-based vhosting](#)
- [More complicated name-based vhosts](#)
- [IP-based vhosts](#)
- [Mixed name-/IP-based vhosts](#)
- [Port-based vhosts](#)

Additional features

- [Using default vhosts](#)
 - [Migrating a named-based vhost to an IP-based vhost](#)
 - [Using the ServerPath directive](#)
-

Simple name-based vhosting

- **Compatibility:** This syntax was added in Apache 1.3.13.
- **Setup:** The server machine has a primary name server.domain.tld. There are two aliases (CNAMEs) www.domain.tld and www.sub.domain.tld for the address server.domain.tld.

Server configuration:

```
...
Port 80
ServerName server.domain.tld

NameVirtualHost *

<VirtualHost *>
DocumentRoot /www/domain
ServerName www.domain.tld
...
</VirtualHost>

<VirtualHost *>
DocumentRoot /www/subdomain
ServerName www.sub.domain.tld
...
</VirtualHost>
```

The asterisks match all addresses, so the main server serves no requests. Due to the fact that `www.domain.tld` is first in the configuration file, it has the highest priority and can be seen as the default or primary server.

More complicated name-based vhosts

- **Setup 1:** The server machine has one IP address (111.22.33.44) which resolves to the name `server.domain.tld`. There are two aliases (CNAMEs) `www.domain.tld` and `www.sub.domain.tld` for the address 111.22.33.44.

Server configuration:

```
...
Port 80
ServerName server.domain.tld

NameVirtualHost 111.22.33.44

<VirtualHost 111.22.33.44>
DocumentRoot /www/domain
ServerName www.domain.tld
...
</VirtualHost>

<VirtualHost 111.22.33.44>
DocumentRoot /www/subdomain
ServerName www.sub.domain.tld
...
</VirtualHost>
```

Apart from localhost there are no unspecified addresses/ports, therefore the main server only serves localhost requests. Due to the fact that `www.domain.tld` has the highest priority it can be seen as the default or primary server.

- **Setup 2:** The server machine has two IP addresses (111.22.33.44 and 111.22.33.55) which resolve to the names `server1.domain.tld` and `server2.domain.tld` respectively. The alias `www.domain.tld` should be used for the main server which should also catch any unspecified addresses. We want to use a virtual host for the alias `www.otherdomain.tld` and another virtual host, with server name `www.sub.domain.tld`, should catch any request to hostnames of the form `*.sub.domain.tld`. The address 111.22.33.55 should be used for the virtual hosts.

Server configuration:

```
...
Port 80
ServerName www.domain.tld
DocumentRoot /www/domain

NameVirtualHost 111.22.33.55

<VirtualHost 111.22.33.55>
DocumentRoot /www/otherdomain
ServerName www.otherdomain.tld
...
</VirtualHost>

<VirtualHost 111.22.33.55>
DocumentRoot /www/subdomain
ServerName www.sub.domain.tld
ServerAlias *.sub.domain.tld
...
</VirtualHost>
```

Any request to an address other than 111.22.33.55 will be served from the main server. A request to 111.22.33.55 with an unknown or no `Host :` header will be served from `www.otherdomain.tld`.

- **Setup 3:** The server machine has two IP addresses (192.168.1.1 and 111.22.33.55). The machine is sitting between an internal (intranet) network and an external (internet) network. Outside of the network, the name `server1.domain.tld` resolves to the external address (111.22.33.55), but inside the network, that same name resolves to the internal address (192.168.1.1).

The server can be made to respond to internal and external requests with the same content, with just one `VirtualHost` section.

Server configuration:

```
...
NameVirtualHost 192.168.1.1
NameVirtualHost 111.22.33.55

<VirtualHost 192.168.1.1 111.22.33.55>
DocumentRoot /www/server1
ServerName server1.domain.tld
ServerAlias server1
...
</VirtualHost>
```

Now requests from both networks will be served from the same `VirtualHost`

- **Setup 4:** You have multiple domains going to the same IP and also want to serve multiple ports. By defining the ports in the "NameVirtualHost" tag, you can allow this to work. If you try using `<VirtualHost name:port>` without the `NameVirtualHost name:port` or you try to use the `Port` directive, your configuration will not work.

Server configuration:

```
...
NameVirtualHost 111.22.33.44:80
NameVirtualHost 111.22.33.44:8080

<VirtualHost 111.22.33.44:80>
ServerName www.domain.tld
DocumentRoot /www/domain-80
</VirtualHost>

<VirtualHost 111.22.33.44:8080>
ServerName www.domain.tld
DocumentRoot /www/domain-8080
</VirtualHost>

<VirtualHost 111.22.33.44:80>
ServerName www.otherdomain.tld
DocumentRoot /www/otherdomain-80
</VirtualHost>

<VirtualHost 111.22.33.44:8080>
ServerName www.otherdomain.tld
DocumentRoot /www/otherdomain-8080
</VirtualHost>
```

IP-based vhosts

- **Setup 1:** The server machine has two IP addresses (111.22.33.44 and 111.22.33.55) which resolve to the names server.domain.tld and www.otherdomain.tld respectively. The hostname www.domain.tld is an alias (CNAME) for server.domain.tld and will represent the main server.

Server configuration:

```
...
Port 80
DocumentRoot /www/domain
ServerName www.domain.tld

<VirtualHost 111.22.33.55>
DocumentRoot /www/otherdomain
ServerName www.otherdomain.tld
...
</VirtualHost>
```

www.otherdomain.tld can only be reached through the address 111.22.33.55, while www.domain.tld can only be reached through 111.22.33.44 (which represents our main server).

- **Setup 2:** Same as setup 1, but we don't want to have a dedicated main server.

Server configuration:

```
...
Port 80
ServerName server.domain.tld

<VirtualHost 111.22.33.44>
DocumentRoot /www/domain
ServerName www.domain.tld
...
</VirtualHost>

<VirtualHost 111.22.33.55>
DocumentRoot /www/otherdomain
ServerName www.otherdomain.tld
...
</VirtualHost>
```

The main server can never catch a request, because all IP addresses of our machine are in use for IP-based virtual hosts (only localhost requests can hit the main server).

- **Setup 3:** The server machine has two IP addresses (111.22.33.44 and 111.22.33.55) which resolve to the names server.domain.tld and www-cache.domain.tld respectively. The hostname www.domain.tld is an alias (CNAME) for server.domain.tld and will represent the main server. www-cache.domain.tld will become our proxy-cache listening on port 8080, while the web server itself uses the default port 80.

Server configuration:

```
...
Port 80
Listen 111.22.33.44:80
Listen 111.22.33.55:8080
ServerName server.domain.tld

<VirtualHost 111.22.33.44:80>
DocumentRoot /www/domain
ServerName www.domain.tld
```

```

...
</VirtualHost>

<VirtualHost 111.22.33.55:8080>
ServerName www-cache.domain.tld
...
  <Directory proxy:>
    Order Deny,Allow
    Deny from all
    Allow from 111.22.33
  </Directory>
</VirtualHost>

```

The main server can never catch a request, because all IP addresses (apart from localhost) of our machine are in use for IP-based virtual hosts. The web server can only be reached on the first address through port 80 and the proxy only on the second address through port 8080.

Mixed name-/IP-based vhosts

- **Setup:** The server machine has three IP addresses (111.22.33.44, 111.22.33.55 and 111.22.33.66) which resolve to the names server.domain.tld, www.otherdomain1.tld and www.otherdomain2.tld respectively. The address 111.22.33.44 should we used for a couple of name-based vhosts and the other addresses for IP-based vhosts.

Server configuration:

```

...
Port 80
ServerName server.domain.tld

NameVirtualHost 111.22.33.44

<VirtualHost 111.22.33.44>
DocumentRoot /www/domain
ServerName www.domain.tld
...
</VirtualHost>

<VirtualHost 111.22.33.44>
DocumentRoot /www/subdomain1
ServerName www.sub1.domain.tld
...
</VirtualHost>

<VirtualHost 111.22.33.44>
DocumentRoot /www/subdomain2
ServerName www.sub2.domain.tld
...
</VirtualHost>

<VirtualHost 111.22.33.55>
DocumentRoot /www/otherdomain1
ServerName www.otherdomain1.tld
...
</VirtualHost>

<VirtualHost 111.22.33.66>
DocumentRoot /www/otherdomain2
ServerName www.otherdomain2.tld
...

```

```
</VirtualHost>
```

Port-based vhosts

- **Setup:** The server machine has one IP address (111.22.33.44) which resolves to the name `www.domain.tld`. If we don't have the option to get another address or alias for our server we can use port-based vhosts if we need a virtual host with a different configuration.

Server configuration:

```
...
Listen 80
Listen 8080
ServerName www.domain.tld
DocumentRoot /www/domain

<VirtualHost 111.22.33.44:8080>
DocumentRoot /www/domain2
...
</VirtualHost>
```

A request to `www.domain.tld` on port 80 is served from the main server and a request to port 8080 is served from the virtual host.

Using `_default_` vhosts

- **Setup 1:** Catching *every* request to any unspecified IP address and port, *i.e.*, an address/port combination that is not used for any other virtual host.

Server configuration:

```
...
<VirtualHost _default_:*>
DocumentRoot /www/default
...
</VirtualHost>
```

Using such a default vhost with a wildcard port effectively prevents any request going to the main server. A default vhost never serves a request that was sent to an address/port that is used for name-based vhosts. If the request contained an unknown or no `Host :` header it is always served from the primary name-based vhost (the vhost for that address/port appearing first in the configuration file).

You can use [AliasMatch](#) or [RewriteRule](#) to rewrite any request to a single information page (or script).

- **Setup 2:** Same as setup 1, but the server listens on several ports and we want to use a second `_default_` vhost for port 80.

Server configuration:

```
...
<VirtualHost _default_:80>
DocumentRoot /www/default80
...
</VirtualHost>

<VirtualHost _default_:*>
```

```

DocumentRoot /www/default
...
</VirtualHost>

```

The default vhost for port 80 (which *must* appear before any default vhost with a wildcard port) catches all requests that were sent to an unspecified IP address. The main server is never used to serve a request.

- **Setup 3:** We want to have a default vhost for port 80, but no other default vhosts.

Server configuration:

```

...
<VirtualHost _default_:80>
DocumentRoot /www/default
...
</VirtualHost>

```

A request to an unspecified address on port 80 is served from the default vhost any other request to an unspecified address and port is served from the main server.

Migrating a name-based vhost to an IP-based vhost

- **Setup:** The name-based vhost with the hostname `www.otherdomain.tld` (from our [name-based](#) example, setup 2) should get its own IP address. To avoid problems with name servers or proxies who cached the old IP address for the name-based vhost we want to provide both variants during a migration phase.

The solution is easy, because we can simply add the new IP address (111.22.33.66) to the `VirtualHost` directive.

Server configuration:

```

...
Port 80
ServerName www.domain.tld
DocumentRoot /www/domain

NameVirtualHost 111.22.33.55

<VirtualHost 111.22.33.55 111.22.33.66>
DocumentRoot /www/otherdomain
ServerName www.otherdomain.tld
...
</VirtualHost>

<VirtualHost 111.22.33.55>
DocumentRoot /www/subdomain
ServerName www.sub.domain.tld
ServerAlias *.sub.domain.tld
...
</VirtualHost>

```

The vhost can now be accessed through the new address (as an IP-based vhost) and through the old address (as a name-based vhost).

Using the ServerPath directive

- **Setup:** We have a server with two name-based vhosts. In order to match the correct virtual host a client must send the correct `Host` header. Old HTTP/1.0 clients do not send such a header and Apache has no clue what vhost the client tried to reach (and serves the request from the primary vhost). To provide as much backward compatibility as possible we create a primary vhost which returns a single page containing links with an URL prefix to the name-based virtual hosts.

Server configuration:

```

...
NameVirtualHost 111.22.33.44

<VirtualHost 111.22.33.44>
# primary vhost
DocumentRoot /www/subdomain
RewriteEngine On
RewriteRule ^/.* /www/subdomain/index.html
...
</VirtualHost>

<VirtualHost 111.22.33.44>
DocumentRoot /www/subdomain/sub1
ServerName www.sub1.domain.tld
ServerPath /sub1/
RewriteEngine On
RewriteRule ^(/sub1/.* ) /www/subdomain$1
...
</VirtualHost>

<VirtualHost 111.22.33.44>
DocumentRoot /www/subdomain/sub2
ServerName www.sub2.domain.tld
ServerPath /sub2/
RewriteEngine On
RewriteRule ^(/sub2/.* ) /www/subdomain$1
...
</VirtualHost>

```

Due to the [ServerPath](#) directive a request to the URL `http://www.sub1.domain.tld/sub1/` is *always* served from the sub1-vhost.

A request to the URL `http://www.sub1.domain.tld/` is only served from the sub1-vhost if the client sent a correct `Host` header. If no `Host` header is sent the client gets the information page from the primary host.

Please note that there is one oddity: A request to `http://www.sub2.domain.tld/sub1/` is also served from the sub1-vhost if the client sent no `Host` header.

The `RewriteRule` directives are used to make sure that a client which sent a correct `Host` header can use both URL variants, *i.e.*, with or without URL prefix.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

An In-Depth Discussion of Virtual Host Matching

The virtual host code was completely rewritten in **Apache 1.3**. This document attempts to explain exactly what Apache does when deciding what virtual host to serve a hit from. With the help of the new [NameVirtualHost](#) directive virtual host configuration should be a lot easier and safer than with versions prior to 1.3.

If you just want to make it work without understanding how, here are [some examples](#).

Config File Parsing

There is a *main_server* which consists of all the definitions appearing outside of `<VirtualHost>` sections. There are virtual servers, called *vhosts*, which are defined by `<VirtualHost>` sections.

The directives [Port](#), [ServerName](#), [ServerPath](#), and [ServerAlias](#) can appear anywhere within the definition of a server. However, each appearance overrides the previous appearance (within that server).

The default value of the `Port` field for *main_server* is 80. The *main_server* has no default `ServerPath`, or `ServerAlias`. The default `ServerName` is deduced from the servers IP address.

The *main_server* `Port` directive has two functions due to legacy compatibility with NCSA configuration files. One function is to determine the default network port Apache will bind to. This default is overridden by the existence of any [Listen](#) directives. The second function is to specify the port number which is used in absolute URIs during redirects.

Unlike the *main_server*, *vhost* ports *do not* affect what ports Apache listens for connections on.

Each address appearing in the `VirtualHost` directive can have an optional port. If the port is unspecified it defaults to the value of the *main_server*'s most recent `Port` statement. The special port `*` indicates a wildcard that matches any port. Collectively the entire set of addresses (including multiple A record results from DNS lookups) are called the *vhost's address set*.

Unless a [NameVirtualHost](#) directive is used for a specific IP address the first *vhost* with that address is treated as an IP-based *vhost*. In 1.3.13 and later that includes the IP address `*`.

If name-based *vhosts* should be used a `NameVirtualHost` directive *must* appear with the IP address set to be used for the name-based *vhosts*. In other words, you must specify the IP address that holds the hostname aliases (CNAMEs) for your name-based *vhosts* via a `NameVirtualHost` directive in your configuration file.

Multiple `NameVirtualHost` directives can be used each with a set of `VirtualHost` directives but only one `NameVirtualHost` directive should be used for each specific IP:port pair.

The ordering of `NameVirtualHost` and `VirtualHost` directives is not important which makes the following two examples identical (only the order of the `VirtualHost` directives for *one* address set is important, see below):

```

NameVirtualHost 111.22.33.44 | <VirtualHost 111.22.33.44>
<VirtualHost 111.22.33.44> | # server A
# server A | </VirtualHost>
... | <VirtualHost 111.22.33.55>
</VirtualHost> | # server C
<VirtualHost 111.22.33.44> | ...
# server B | </VirtualHost>

```

```

...           | <VirtualHost 111.22.33.44>
</VirtualHost> | # server B
                | ...
NameVirtualHost 111.22.33.55 | </VirtualHost>
<VirtualHost 111.22.33.55> | <VirtualHost 111.22.33.55>
# server C           | # server D
...                 | ...
</VirtualHost>     | </VirtualHost>
<VirtualHost 111.22.33.55> |
# server D           | NameVirtualHost 111.22.33.44
...                 | NameVirtualHost 111.22.33.55
</VirtualHost>     |

```

(To aid the readability of your configuration you should prefer the left variant.)

After parsing the `VirtualHost` directive, the `vhost` server is given a default `Port` equal to the port assigned to the first name in its `VirtualHost` directive.

The complete list of names in the `VirtualHost` directive are treated just like a `ServerAlias` (but are not overridden by any `ServerAlias` statement) if all names resolve to the same address set. Note that subsequent `Port` statements for this `vhost` will not affect the ports assigned in the address set.

During initialization a list for each IP address is generated and inserted into an hash table. If the IP address is used in a `NameVirtualHost` directive the list contains all name-based `vhosts` for the given IP address. If there are no `vhosts` defined for that address the `NameVirtualHost` directive is ignored and an error is logged. For an IP-based `vhost` the list in the hash table is empty.

Due to a fast hashing function the overhead of hashing an IP address during a request is minimal and almost not existent. Additionally the table is optimized for IP addresses which vary in the last octet.

For every `vhost` various default values are set. In particular:

1. If a `vhost` has no [ServerAdmin](#), [ResourceConfig](#), [AccessConfig](#), [Timeout](#), [KeepAliveTimeout](#), [KeepAlive](#), [MaxKeepAliveRequests](#), or [SendBufferSize](#) directive then the respective value is inherited from the `main_server`. (That is, inherited from whatever the final setting of that value is in the `main_server`.)
2. The "lookup defaults" that define the default directory permissions for a `vhost` are merged with those of the `main_server`. This includes any per-directory configuration information for any module.
3. The per-server configs for each module from the `main_server` are merged into the `vhost` server.

Essentially, the `main_server` is treated as "defaults" or a "base" on which to build each `vhost`. But the positioning of these `main_server` definitions in the config file is largely irrelevant -- the entire config of the `main_server` has been parsed when this final merging occurs. So even if a `main_server` definition appears after a `vhost` definition it might affect the `vhost` definition.

If the `main_server` has no `ServerName` at this point, then the hostname of the machine that `httpd` is running on is used instead. We will call the *main_server address set* those IP addresses returned by a DNS lookup on the `ServerName` of the `main_server`.

For any undefined `ServerName` fields, a name-based `vhost` defaults to the address given first in the `VirtualHost` statement defining the `vhost`.

Any `vhost` that includes the magic `_default_` wildcard is given the same `ServerName` as the `main_server`.

Virtual Host Matching

The server determines which `vhost` to use for a request as follows:

Hash table lookup

When the connection is first made by a client, the IP address to which the client connected is looked up in the internal IP hash table.

If the lookup fails (the IP address wasn't found) the request is served from the `_default_` `vhost` if there is such a `vhost` for the port to which the client sent the request. If there is no matching `_default_` `vhost` the request is served from the `main_server`.

In Apache 1.3.13 and later, if the IP address is not found in the hash table then the match against the port number may also result in an entry corresponding to a `NameVirtualHost *`, which is subsequently handled like other name-based vhosts.

If the lookup succeeded (a corresponding list for the IP address was found) the next step is to decide if we have to deal with an IP-based or a name-base vhost.

IP-based vhost

If the entry we found has an empty name list then we have found an IP-based vhost, no further actions are performed and the request is served from that vhost.

Name-based vhost

If the entry corresponds to a name-based vhost the name list contains one or more vhost structures. This list contains the vhosts in the same order as the `VirtualHost` directives appear in the config file.

The first vhost on this list (the first vhost in the config file with the specified IP address) has the highest priority and catches any request to an unknown server name or a request without a `Host :` header field.

If the client provided a `Host :` header field the list is searched for a matching vhost and the first hit on a `ServerName` or `ServerAlias` is taken and the request is served from that vhost. A `Host :` header field can contain a port number, but Apache always matches against the real port to which the client sent the request.

If the client submitted a HTTP/1.0 request without `Host :` header field we don't know to what server the client tried to connect and any existing `ServerPath` is matched against the URI from the request. The first matching path on the list is used and the request is served from that vhost.

If no matching vhost could be found the request is served from the first vhost with a matching port number that is on the list for the IP to which the client connected (as already mentioned before).

Persistent connections

The IP lookup described above is only done *once* for a particular TCP/IP session while the name lookup is done on *every* request during a `KeepAlive`/persistent connection. In other words a client may request pages from different name-based vhosts during a single persistent connection.

Absolute URI

If the URI from the request is an absolute URI, and its hostname and port match the main server or one of the configured virtual hosts *and* match the address and port to which the client sent the request, then the scheme/hostname/port prefix is stripped off and the remaining relative URI is served by the corresponding main server or virtual host. If it does not match, then the URI remains untouched and the request is taken to be a proxy request.

Observations

- A name-based vhost can never interfere with an IP-base vhost and vice versa. IP-based vhosts can only be reached through an IP address of its own address set and never through any other address. The same applies to name-based vhosts, they can only be reached through an IP address of the corresponding address set which must be defined with a `NameVirtualHost` directive.
- `ServerAlias` and `ServerPath` checks are never performed for an IP-based vhost.
- The order of name-/IP-based, the `_default_` vhost and the `NameVirtualHost` directive within the config file is not important. Only the ordering of name-based vhosts for a specific address set is significant. The one name-based vhosts that comes first in the configuration file has the highest priority for its corresponding address set.
- For security reasons the port number given in a `Host :` header field is never used during the matching process. Apache always uses the real port to which the client sent the request.
- If a `ServerPath` directive exists which is a prefix of another `ServerPath` directive that appears later in the configuration file, then the former will always be matched and the latter will never be matched. (That is assuming that no

Host : header field was available to disambiguate the two.)

- If two IP-based vhosts have an address in common, the vhost appearing first in the config file is always matched. Such a thing might happen inadvertently. The server will give a warning in the error logfile when it detects this.
- A `_default_` vhost catches a request only if there is no other vhost with a matching IP address *and* a matching port number for the request. The request is only caught if the port number to which the client sent the request matches the port number of your `_default_` vhost which is your standard `Port` by default. A wildcard port can be specified (*i.e.*, `_default_*`) to catch requests to any available port. In Apache 1.3.13 and later this also applies to `NameVirtualHost *` vhosts.
- The `main_server` is only used to serve a request if the IP address and port number to which the client connected is unspecified and does not match any other vhost (including a `_default_` vhost). In other words the `main_server` only catches a request for an unspecified address/port combination (unless there is a `_default_` vhost which matches that port).
- A `_default_` vhost or the `main_server` is *never* matched for a request with an unknown or missing `Host :` header field if the client connected to an address (and port) which is used for name-based vhosts, *e.g.*, in a `NameVirtualHost` directive.
- You should never specify DNS names in `VirtualHost` directives because it will force your server to rely on DNS to boot. Furthermore it poses a security threat if you do not control the DNS for all the domains listed. There's [more information](#) available on this and the next two topics.
- `ServerName` should always be set for each vhost. Otherwise A DNS lookup is required for each vhost.

Tips

In addition to the tips on the [DNS Issues](#) page, here are some further tips:

- Place all `main_server` definitions before any `VirtualHost` definitions. (This is to aid the readability of the configuration -- the post-config merging process makes it non-obvious that definitions mixed in around virtual hosts might affect all virtual hosts.)
- Group corresponding `NameVirtualHost` and `VirtualHost` definitions in your configuration to ensure better readability.
- Avoid `ServerPaths` which are prefixes of other `ServerPaths`. If you cannot avoid this then you have to ensure that the longer (more specific) prefix vhost appears earlier in the configuration file than the shorter (less specific) prefix (*i.e.*, "`ServerPath /abc`" should appear after "`ServerPath /abc/def`").

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

File Descriptor Limits

When using a large number of Virtual Hosts, Apache may run out of available file descriptors (sometimes called file handles if each Virtual Host specifies different log files. The total number of file descriptors used by Apache is one for each distinct error log file, one for every other log file directive, plus 10-20 for internal use. Unix operating systems limit the number of file descriptors that may be used by a process; the limit is typically 64, and may usually be increased up to a large hard-limit.

Although Apache attempts to increase the limit as required, this may not work if:

1. Your system does not provide the `setrlimit()` system call.
2. The `setrlimit(RLIMIT_NOFILE)` call does not function on your system (such as Solaris 2.3)
3. The number of file descriptors required exceeds the hard limit.
4. Your system imposes other limits on file descriptors, such as a limit on stdio streams only using file descriptors below 256. (Solaris 2)

In the event of problems you can:

- Reduce the number of log files; don't specify log files in the VirtualHost sections, but only log to the main log files.
- If your system falls into 1 or 2 (above), then increase the file descriptor limit before starting Apache, using a script like

```
#!/bin/sh
ulimit -S -n 100
exec httpd
```

Please see the [Descriptors and Apache](#) document containing further details about file descriptor problems and how they can be solved on your operating system.

Apache HTTP Server Version 1.3





Apache HTTP Server

Issues Regarding DNS and Apache

This page could be summarized with the statement: *don't require Apache to use DNS for any parsing of the configuration files*. If Apache has to use DNS to parse the configuration files then your server may be subject to reliability problems (it might not boot), or denial and theft of service attacks (including users able to steal hits from other users).

A Simple Example

Consider this configuration snippet:

```
<VirtualHost www.abc.dom>
ServerAdmin webgirl@abc.dom
DocumentRoot /www/abc
</VirtualHost>
```

In order for Apache to function properly it absolutely needs to have two pieces of information about each virtual host: the [ServerName](#) and at least one IP address that the server responds to. This example does not include the IP address, so Apache must use DNS to find the address of `www.abc.dom`. If for some reason DNS is not available at the time your server is parsing its config file, then this virtual host **will not be configured**. It won't be able to respond to any hits to this virtual host (prior to Apache version 1.2 the server would not even boot).

Suppose that `www.abc.dom` has address 10.0.0.1. Then consider this configuration snippet:

```
<VirtualHost 10.0.0.1>
ServerAdmin webgirl@abc.dom
DocumentRoot /www/abc
</VirtualHost>
```

Now Apache needs to use reverse DNS to find the `ServerName` for this virtualhost. If that reverse lookup fails then it will partially disable the virtualhost (prior to Apache version 1.2 the server would not even boot). If the virtual host is name-based then it will effectively be totally disabled, but if it is IP-based then it will mostly work. However if Apache should ever have to generate a full URL for the server which includes the server name then it will fail to generate a valid URL.

Here is a snippet that avoids both of these problems.

```
<VirtualHost 10.0.0.1>
ServerName www.abc.dom
ServerAdmin webgirl@abc.dom
DocumentRoot /www/abc
</VirtualHost>
```

Denial of Service

There are (at least) two forms that denial of service can come in. If you are running a version of Apache prior to version 1.2 then your server will not even boot if one of the two DNS lookups mentioned above fails for any of your virtual hosts. In some cases this DNS lookup may not even be under your control. For example, if `abc.dom` is one of your customers and they control their own DNS then they can force your (pre-1.2) server to fail while booting simply by deleting the `www.abc.dom` record.

Another form is far more insidious. Consider this configuration snippet:

```
<VirtualHost www.abc.dom>  
ServerAdmin webgirl@abc.dom  
DocumentRoot /www/abc  
</VirtualHost>
```

```
<VirtualHost www.def.dom>  
ServerAdmin webguy@def.dom  
DocumentRoot /www/def  
</VirtualHost>
```

Suppose that you've assigned 10.0.0.1 to `www.abc.dom` and 10.0.0.2 to `www.def.dom`. Furthermore, suppose that `def.com` has control of their own DNS. With this config you have put `def.com` into a position where they can steal all traffic destined to `abc.com`. To do so, all they have to do is set `www.def.dom` to 10.0.0.1. Since they control their own DNS you can't stop them from pointing the `www.def.com` record wherever they wish.

Requests coming in to 10.0.0.1 (including all those where users typed in URLs of the form `http://www.abc.dom/whatever`) will all be served by the `def.com` virtual host. To better understand why this happens requires a more in-depth discussion of how Apache matches up incoming requests with the virtual host that will serve it. A rough document describing this [is available](#).

The "main server" Address

The addition of [name-based virtual host support](#) in Apache 1.1 requires Apache to know the IP address(es) of the host that httpd is running on. To get this address it uses either the global `ServerName` (if present) or calls the C function `gethostname` (which should return the same as typing "hostname" at the command prompt). Then it performs a DNS lookup on this address. At present there is no way to avoid this lookup.

If you fear that this lookup might fail because your DNS server is down then you can insert the hostname in `/etc/hosts` (where you probably already have it so that the machine can boot properly). Then ensure that your machine is configured to use `/etc/hosts` in the event that DNS fails. Depending on what OS you are using this might be accomplished by editing `/etc/resolv.conf`, or maybe `/etc/nsswitch.conf`.

If your server doesn't have to perform DNS for any other reason then you might be able to get away with running Apache with the `HOSTRESORDER` environment variable set to "local". This all depends on what OS and resolver libraries you are using. It also affects CGIs unless you use [mod_env](#) to control the environment. It's best to consult the man pages or FAQs for your OS.

Tips to Avoid these problems

- use IP addresses in `<VirtualHost>`
- use IP addresses in `Listen`
- use IP addresses in `BindAddress`
- ensure all virtual hosts have an explicit `ServerName`
- create a `<VirtualHost _default_*>` server that has no pages to serve

Appendix: Future Directions

The situation regarding DNS is highly undesirable. For Apache 1.2 we've attempted to make the server at least continue booting in the event of failed DNS, but it might not be the best we can do. In any event requiring the use of explicit IP addresses in configuration files is highly undesirable in today's Internet where renumbering is a necessity.

A possible work around to the theft of service attack described above would be to perform a reverse DNS lookup on the ip address returned by the forward lookup and compare the two names. In the event of a mismatch the virtualhost would be disabled. This would require reverse DNS to be configured properly (which is something that most admins are familiar with because of the common use of "double-reverse" DNS lookups by FTP servers and TCP wrappers).

In any event it doesn't seem possible to reliably boot a virtual-hosted web server when DNS has failed unless IP addresses are used. Partial solutions such as disabling portions of the configuration might be worse than not booting at all depending on what the webserver is supposed to accomplish.

As HTTP/1.1 is deployed and browsers and proxies start issuing the `Host` header it will become possible to avoid the use of IP-based virtual hosts entirely. In this event a webserver has no requirement to do DNS lookups during configuration. But as of March 1997 these features have not been deployed widely enough to be put into use on critical web servers.

Apache HTTP Server





Apache HTTP Server Version 1.3

Apache Server Frequently Asked Questions

\$Revision: 1.149 \$ (\$Date: 2001/10/08 01:26:54 \$)

The latest version of this FAQ is always available from the main Apache web site, at <http://httpd.apache.org/docs/misc/FAQ.html>.

If you are reading a text-only version of this FAQ, you may find numbers enclosed in brackets (such as "[12]"). These refer to the list of reference URLs to be found at the end of the document. These references do not appear, and are not needed, for the hypertext version.

The Questions

. Background

1. [What is Apache?](#)
2. [How and why was Apache created?](#)
3. [Why the name "Apache"?](#)
4. [OK, so how does Apache compare to other servers?](#)
5. [How thoroughly tested is Apache?](#)
6. [What are the future plans for Apache?](#)
7. [Whom do I contact for support?](#)
8. [Is there any more information on Apache?](#)
9. [Where can I get Apache?](#)
10. [May I use the Apache logo on my product or Web site?](#)

B. General Technical Questions

1. ["Why can't I ...? Why won't ... work?" What to do in case of problems](#)
2. [How compatible is Apache with my existing NCSA 1.3 setup?](#)
3. [Is Apache Year 2000 compliant?](#)
4. [How do I submit a patch to the Apache Group?](#)
5. [Why has Apache stolen my favourite site's Internet address?](#)
6. [Why am I getting spam mail from the Apache site?](#)
7. [May I include the Apache software on a CD or other package I'm distributing?](#)
8. [What's the best hardware/operating system/... How do I get the most out of my Apache Web server?](#)
9. [What are "regular expressions"?](#)
10. [Why isn't there a binary for my platform?](#)

C. Building Apache

1. [Why do I get an error about an undefined reference to "`inet_ntoa`" or other `inet_*` symbols?](#)
2. [Why won't Apache compile with my system's cc?](#)
3. [Why do I get complaints about redefinition of "`struct iovec`" when compiling under Linux?](#)
4. [I'm using gcc and I get some compilation errors, what is wrong?](#)

5. [I'm using RedHat Linux 5.0, or some other glibc-based Linux system, and I get errors with the `crypt` function when I attempt to build Apache 1.2.](#)

D. Error Log Messages and Problems Starting Apache

1. [Why do I get "setgid: Invalid argument" at startup?](#)
2. [Why am I getting "httpd: could not set socket option TCP_NODELAY" in my error log?](#)
3. [Why am I getting "connection reset by peer" in my error log?](#)
4. [The errorlog says Apache dumped core, but where's the dump file?](#)
5. [When I run it under Linux I get "shmget: function not found", what should I do?](#)
6. [Server hangs, or fails to start, and/or error log fills with "fcntl: F_SETLKW: No record locks available" or similar messages](#)
7. [Why am I getting "Expected </Directory> but saw </Directory>" when I try to start Apache?](#)
8. [I'm using RedHat Linux and I have problems with httpd dying randomly or not restarting properly](#)
9. [I upgraded from an Apache version earlier than 1.2.0 and suddenly I have problems with Apache dying randomly or not restarting properly](#)
10. [When I try to start Apache from a DOS window, I get a message like "Cannot determine host name. Use ServerName directive to set it manually." What does this mean?](#)
11. [When I try to start Apache for Windows, I get a message like "Unable To Locate WS2_32.DLL...". What should I do?](#)
12. [Apache for Windows does not start. Error log contains this message "\[crit\] \(10045\) The attempted operation is not supported for the type of object referenced: Parent: WSADuplicateSocket failed for socket ###". What does this mean?](#)
13. [When I try to start Apache on Windows, I get a message like "System error 1067 has occurred. The process terminated unexpectedly." What does this mean?](#)
14. [On a SuSE Linux system, I try and configure access control using basic authentication. Although I follow the example exactly, authentication fails, and an error message "admin: not a valid FDN:" is logged.](#)

E. Configuration Questions

1. [Why can't I run more than `<n>` virtual hosts?](#)
2. [Can I increase `FD_SETSIZE` on FreeBSD?](#)
3. [Why doesn't my `ErrorDocument 401` work?](#)
4. [Why does Apache send a cookie on every response?](#)
5. [Why don't my cookies work, I even compiled in `mod_cookies`?](#)
6. [Why do my Java app\[let\]s give me plain text when I request an URL from an Apache server?](#)
7. [How do I get Apache to send a MIDI file so the browser can play it?](#)
8. [How do I add browsers and referrers to my logs?](#)
9. [Why does accessing directories only work when I include the trailing "/" \(e.g., `http://foo.domain.com/~user/`\) but not when I omit it \(e.g., `http://foo.domain.com/~user`\)?](#)
10. [Why doesn't `mod_info` list any directives?](#)
11. [I upgraded to Apache 1.3 and now my virtual hosts don't work!](#)
12. [I'm using RedHat Linux and my `.htm` files are showing up as HTML source rather than being formatted!](#)
13. [My `.htaccess` files are being ignored.](#)
14. [Why do I get a "Forbidden" message whenever I try to access a particular directory?](#)
15. [Why do I get a "Forbidden/You don't have permission to access / on this server" message whenever I try to access my server?](#)
16. [Why do my files appear correctly in Internet Explorer, but show up as source or trigger a save window with Netscape?](#)
17. [My site is accessible under many different hostnames; how do I redirect clients so that they see only a single name?](#)

F. Dynamic Content (CGI and SSI)

1. [How do I enable CGI execution in directories other than the ScriptAlias?](#)
2. [What does it mean when my CGIs fail with "Premature end of script headers"?](#)
3. [Why do I keep getting "Method Not Allowed" for form POST requests?](#)
4. [How can I get my script's output without Apache buffering it? Why doesn't my server push work?](#)
5. [Where can I find the "CGI specification"?](#)
6. [Why isn't FastCGI included with Apache any more?](#)
7. [How do I enable SSI \(parsed HTML\)?](#)
8. [Why don't my parsed files get cached?](#)
9. [How can I have my script output parsed?](#)
10. [SSIs don't work for VirtualHosts and/or user home directories](#)
11. [How can I use `ErrorDocument` and SSI to simplify customized error messages?](#)
12. [Why is the environment variable `REMOTE_USER` not set?](#)
13. [How do I allow each of my user directories to have a cgi-bin directory?](#)

G. Authentication and Access Restrictions

1. [Why isn't restricting access by host or domain name working correctly?](#)
2. [How do I set up Apache to require a username and password to access certain documents?](#)
3. [How do I set up Apache to allow access to certain documents only if a site is either a local site *or* the user supplies a password and username?](#)
4. [Why does my authentication give me a server error?](#)
5. [Do I have to keep the \(mSQL\) authentication information on the same machine?](#)
6. [Why is my mSQL authentication terribly slow?](#)
7. [Can I use my `/etc/passwd` file for Web page authentication?](#)
8. [Why does Apache ask for my password twice before serving a file?](#)

H. URL Rewriting

1. [Where can I find `mod_rewrite` rule sets which already solve particular URL-related problems?](#)
2. [Where can I find any published information about URL-manipulations and `mod_rewrite`?](#)
3. [Why is `mod_rewrite` so difficult to learn and seems so complicated?](#)
4. [What can I do if my `RewriteRules` don't work as expected?](#)
5. [Why don't some of my URLs get prefixed with `DocumentRoot` when using `mod_rewrite`?](#)
6. [How can I make all my URLs case-insensitive with `mod_rewrite`?](#)
7. [Why are `RewriteRules` in my `VirtualHost` parts ignored?](#)
8. [How can I use strings with whitespaces in `RewriteRule`'s `ENV` flag?](#)

I. Features

1. [Does or will Apache act as a Proxy server?](#)
2. [What are "multiviews"?](#)
3. [Why can't I publish to my Apache server using PUT on Netscape Gold and other programs?](#)
4. [Why doesn't Apache include SSL?](#)
5. [How can I attach a footer to my documents without using SSI?](#)
6. [Does Apache include a search engine?](#)
7. [How can I rotate my log files?](#)
8. [How do I keep certain requests from appearing in my logs?](#)

9. [Does Apache include any sort of database integration?](#)
 10. [Can I use Active Server Pages \(ASP\) with Apache?](#)
 11. [Does Apache come with Java support?](#)
-

The Answers

A. Background

1. What is Apache?

The Apache httpd server

- is a powerful, flexible, HTTP/1.1 compliant web server
- implements the latest protocols, including HTTP/1.1 (RFC2616)
- is highly configurable and extensible with third-party modules
- can be customised by writing 'modules' using the Apache module API
- provides full source code and comes with an unrestrictive license
- runs on Windows NT/9x, Netware 5.x, OS/2, and most versions of Unix, as well as several other operating systems
- is actively being developed
- encourages user feedback through new ideas, bug reports and patches
- implements many frequently requested features, including:

DBM databases for authentication

allows you to easily set up password-protected pages with enormous numbers of authorized users, without bogging down the server.

Customized responses to errors and problems

Allows you to set up files, or even CGI scripts, which are returned by the server in response to errors and problems, e.g. setup a script to intercept **500 Server Errors** and perform on-the-fly diagnostics for both users and yourself.

Multiple DirectoryIndex directives

Allows you to say `DirectoryIndex index.html index.cgi`, which instructs the server to either send back `index.html` or run `index.cgi` when a directory URL is requested, whichever it finds in the directory.

Unlimited flexible URL rewriting and aliasing

Apache has no fixed limit on the numbers of Aliases and Redirects which may be declared in the config files. In addition, a powerful rewriting engine can be used to solve most URL manipulation problems.

Content negotiation

i.e. the ability to automatically serve clients of varying sophistication and HTML level compliance, with documents which offer the best representation of information that the client is capable of accepting.

Virtual Hosts

A much requested feature, sometimes known as multi-homed servers. This allows the server to distinguish between requests made to different IP addresses or names (mapped to the same machine). Apache also offers dynamically configurable mass-virtual hosting.

Configurable Reliable Piped Logs

You can configure Apache to generate logs in the format that you want. In addition, on most Unix architectures, Apache can send log files to a pipe, allowing for log rotation, hit filtering, real-time splitting of multiple vhosts into separate logs, and asynchronous DNS resolving on the fly.

2. How and why was Apache created?

The [About Apache](#) document explains how the Apache project evolved from its beginnings as an outgrowth of the NCSA httpd project to its current status as one of the fastest, most efficient, and most functional web servers in existence.

3. Why the name "Apache"?

A cute name which stuck. Apache is "A PAtCHy server". It was based on some existing code and a series of "patch files".

For many developers it is also a reverent connotation to the Native American Indian tribe of Apache, [well-known for their superior skills in warfare strategy and inexhaustible endurance](#). For more information on the Apache Nation, we suggest searching [Google](#), [Northernlight](#), or [AllTheWeb](#).

4. OK, so how does Apache compare to other servers?

For an independent assessment, see [Web Compare](#).

Apache has been shown to be substantially faster, more stable, and more feature-full than many other web servers. Although certain commercial servers have claimed to surpass Apache's speed (it has not been demonstrated that any of these "benchmarks" are a good way of measuring WWW server speed at any rate), we feel that it is better to have a mostly-fast free server than an extremely-fast server that costs thousands of dollars. Apache is run on sites that get millions of hits per day, and they have experienced no performance difficulties.

5. How thoroughly tested is Apache?

Apache is run on over 6 million Internet servers (as of February 2000). It has been tested thoroughly by both developers and users. The Apache Group maintains rigorous standards before releasing new versions of their server, and our server runs without a hitch on over one half of all WWW servers available on the Internet. When bugs do show up, we release patches and new versions as soon as they are available.

6. What are the future plans for Apache?

- to continue to be an "open source" no-charge-for-use HTTP server,
 - to keep up with advances in HTTP protocol and web developments in general,
 - to collect suggestions for fixes/improvements from its users,
 - to respond to needs of large volume providers as well as occasional users.
-

7. Whom do I contact for support?

There is no official support for Apache. None of the developers want to be swamped by a flood of trivial questions that can be resolved elsewhere. Bug reports and suggestions should be sent *via* [the bug report page](#). Other questions should be directed to the [Apache HTTP Server Users List](#) or the [comp.infosystems.www.servers.unix](#) or [comp.infosystems.www.servers.ms-windows](#) newsgroup (as appropriate for the platform you use), where some of the Apache team lurk, in the company of many other httpd gurus who should be able to help.

Commercial support for Apache is, however, available from a number of third parties.

8. Is there any more information available on Apache?

Indeed there is. See the main [Apache web site](#). There is also a regular electronic publication called [Apache Week](#) available. Links to relevant Apache Week articles are included below where appropriate. There are also some [Apache-specific books](#) available.

9. Where can I get Apache?

You can find out how to download the source for Apache at the project's [main web page](#).

10. May I use the Apache logo on my product or Web site?

You may **NOT** use any original artwork from the Apache Software Foundation, nor make or use modified versions of such artwork, except under the following conditions:

- You may use the ['Powered by Apache' graphic](#) on a Web site that is being served by the Apache HTTP server software.
- You may use the aforementioned 'Powered by Apache' graphic or the [Apache Software Foundation logo](#) in product

description and promotional material **IF and ONLY IF** such use can in no way be interpreted as anything other than an attribution. Using the Apache name and artwork in a manner that implies endorsement of a product or service is **strictly forbidden**.

B. General Technical Questions

1. "Why can't I ...? Why won't ... work?" What to do in case of problems

If you are having trouble with your Apache server software, you should take the following steps:

1. Check the errorlog!

Apache tries to be helpful when it encounters a problem. In many cases, it will provide some details by writing one or messages to the server error log. Sometimes this is enough for you to diagnose & fix the problem yourself (such as file permissions or the like). The default location of the error log is `/usr/local/apache/logs/error_log`, but see the [ErrorLog](#) directive in your config files for the location on your server.

2. Check the [FAQ](#)!

The latest version of the Apache Frequently-Asked Questions list can always be found at the main Apache web site.

3. Check the Apache bug database

Most problems that get reported to The Apache Group are recorded in the [bug database](#). *Please check the existing reports, open **and** closed, before adding one.* If you find that your issue has already been reported, please *don't* add a "me, too" report. If the original report isn't closed yet, we suggest that you check it periodically. You might also consider contacting the original submitter, because there may be an email exchange going on about the issue that isn't getting recorded in the database.

4. Ask in a user support group.

A lot of common problems never make it to the bug database because there's already high Q&A traffic about them in the [Users mailing list](#) or [comp.infosystems.www.servers.unix](#) and related newsgroups. These newsgroups are also available via [Google](#). Many Apache users, and some of the developers, can be found roaming their virtual halls, so it is suggested that you seek wisdom there. The chances are good that you'll get a faster answer there than from the bug database, even if you *don't* see your question already posted.

5. If all else fails, report the problem in the bug database

If you've gone through those steps above that are appropriate and have obtained no relief, then please *do* let The Apache Group know about the problem by [logging a bug report](#).

If your problem involves the server crashing and generating a core dump, please include a backtrace (if possible). As an example,

```
# cd ServerRoot
# dbx httpd core
(dbx) where
```

(Substitute the appropriate locations for your ServerRoot and your httpd and core files. You may have to use `gdb` instead of `dbx`.)

2. How compatible is Apache with my existing NCSA 1.3 setup?

Apache attempts to offer all the features and configuration options of NCSA httpd 1.3, as well as many of the additional features found in NCSA httpd 1.4 and NCSA httpd 1.5.

NCSA httpd appears to be moving toward adding experimental features which are not generally required at the moment. Some of the experiments will succeed while others will inevitably be dropped. The Apache philosophy is to add what's needed as and when it is needed.

Friendly interaction between Apache and NCSA developers should ensure that fundamental feature enhancements stay consistent between the two servers for the foreseeable future.

3. Is Apache Year 2000 compliant?

Yes, Apache is Year 2000 compliant.

Apache internally never stores years as two digits. On the HTTP protocol level RFC1123-style addresses are generated which is the only format a HTTP/1.1-compliant server should generate. To be compatible with older applications Apache recognizes ANSI C's `asctime()` and RFC850-/RFC1036-style date formats, too. The `asctime()` format uses four-digit years, but the RFC850 and RFC1036 date formats only define a two-digit year. If Apache sees such a date with a value less than 70 it assumes that the century is 20 rather than 19.

Although Apache is Year 2000 compliant, you may still get problems if the underlying OS has problems with dates past year 2000 (e.g., OS calls which accept or return year numbers). Most (UNIX) systems store dates internally as signed 32-bit integers which contain the number of seconds since 1st January 1970, so the magic boundary to worry about is the year 2038 and not 2000. But modern operating systems shouldn't cause any trouble at all.

Users of Apache 1.2.x should upgrade to a current version of Apache 1.3 (see [year-2000 improvements in Apache 1.3](#) for details).

The Apache HTTP Server project is an open-source software product of the Apache Software Foundation. The project and the Foundation **cannot** offer legal assurances regarding any suitability of the software for your application. There are several commercial Apache support organizations and derivative server products available that may be able to stand behind the software and provide you with any assurances you may require. You may find links to some of these vendors at <http://www.apache.org/info/support.cgi>.

The Apache HTTP server software is distributed with the following disclaimer, found in the software license:

```
THIS SOFTWARE IS PROVIDED BY THE APACHE GROUP ``AS IS'' AND ANY
EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE GROUP OR
ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
OF THE POSSIBILITY OF SUCH DAMAGE.
```

4. How do I submit a patch to the Apache Group?

The Apache Group encourages patches from outside developers. There are 2 main "types" of patches: small bugfixes and general improvements. Bugfixes should be submitting using the Apache [bug report page](#). Improvements, modifications, and additions should follow the instructions below.

In general, the first course of action is to be a member of the `dev@httpd.apache.org` mailing list. This indicates to the Group that you are closely following the latest Apache developments. Your patch file should be generated using either `diff -c` or `diff -u` against the latest CVS tree. To submit your patch, send email to `dev@httpd.apache.org` with a Subject: line that starts with [PATCH] and includes a general description of the patch. In the body of the message, the patch should be clearly described and then included at the end of the message. If the patch-file is long, you can note a URL to the file instead of the file itself. Use of MIME enclosures/attachments should be avoided.

Be prepared to respond to any questions about your patches and possibly defend your code. If your patch results in a lot of discussion, you may be asked to submit an updated patch that incorporate all changes and suggestions.

5. Why has Apache stolen my favourite site's Internet address?

The simple answer is: "It hasn't." This misconception is usually caused by the site in question having migrated to the Apache Web server software, but not having migrated the site's content yet. When Apache is installed, the default page that gets installed tells the Webmaster the installation was successful. The expectation is that this default page will be replaced with the site's real content. If it doesn't, complain to the Webmaster, not to the Apache project -- we just make the software and aren't responsible for what people do (or don't do) with it.

6. Why am I getting spam mail from the Apache site?

The short answer is: "You aren't." Usually when someone thinks the Apache site is originating spam, it's because they've traced the spam to a Web site, and the Web site says it's using Apache. See the [previous FAQ entry](#) for more details on this phenomenon.

No marketing spam originates from the Apache site. The only mail that comes from the site goes only to addresses that have been *requested* to receive the mail.

7. May I include the Apache software on a CD or other package I'm distributing?

The detailed answer to this question can be found in the Apache license, which is included in the Apache distribution in the file LICENSE. You can also find it on the Web at <http://www.apache.org/LICENSE.txt>.

8. What's the best hardware/operating system/... How do I get the most out of my Apache Web server?

Check out Dean Gaudet's [performance tuning page](#).

9. What are "regular expressions"?

Regular expressions are a way of describing a pattern - for example, "all the words that begin with the letter A" or "every 10-digit phone number" or even "Every sentence with two commas in it, and no capital letter Q". Regular expressions (aka "regex"s) are useful in Apache because they let you apply certain attributes against collections of files or resources in very flexible ways - for example, all .gif and .jpg files under any "images" directory could be written as `^/images/.*(jpg|gif)$/`.

The best overview around is probably the one which comes with Perl. We implement a simple subset of Perl's regex support, but it's still a good way to learn what they mean. You can start by going to the [CPAN page on regular expressions](#), and branching out from there.

10. Why isn't there a binary for my platform?

The developers make sure that the software builds and works correctly on the platforms available to them; this does *not* necessarily mean that *your* platform is one of them. In addition, the Apache HTTP server project is primarily source oriented, meaning that distributing valid and buildable source code is the purpose of a release, not making sure that there is a binary package for all of the supported platforms.

If you don't see a kit for your platform listed in the binary distribution area ([URL:http://httpd.apache.org/dist/httpd/binaries/](http://httpd.apache.org/dist/httpd/binaries/)), it means either that the platform isn't available to any of the developers, or that they just haven't gotten around to preparing a binary for it. As this is a voluntary project, they are under no obligation to do so. Users are encouraged and expected to build the software themselves.

The sole exception to these practices is the Windows package. Unlike most Unix and Unix-like platforms, Windows systems do not come with a bundled software development environment, so we *do* prepare binary kits for Windows when we make a release. Again, however, it's a voluntary thing and only a limited number of the developers have the capability to build the InstallShield package, so the Windows release may lag somewhat behind the source release. This lag should be no more than a few days at most.

C. Building Apache

1. Why do I get an error about an undefined reference to "__inet_ntoa" or other __inet_* symbols?

If you have installed [BIND-8](#) then this is normally due to a conflict between your include files and your libraries. BIND-8 installs its include files and libraries `/usr/local/include/` and `/usr/local/lib/`, while the resolver that comes with your system is probably installed in `/usr/include/` and `/usr/lib/`. If your system uses the header files in `/usr/local/include/` before those in `/usr/include/` but you do not use the new resolver library, then the two versions will conflict.

To resolve this, you can either make sure you use the include files and libraries that came with your system or make sure to use the new include files and libraries. Adding `-lbind` to the `EXTRA_LDFLAGS` line in your Configuration file, then re-running Configure, should resolve the problem. (Apache versions 1.2.* and earlier use `EXTRA_LFLAGS` instead.)

Note:As of BIND 8.1.1, the bind libraries and files are installed under `/usr/local/bind` by default, so you should not run into this

problem. Should you want to use the bind resolvers you'll have to add the following to the respective lines:

```
EXTRA_CFLAGS=-I/usr/local/bind/include
EXTRA_LDFLAGS=-L/usr/local/bind/lib
EXTRA_LIBS=-lbind
```

2. Why won't Apache compile with my system's cc?

If the server won't compile on your system, it is probably due to one of the following causes:

- **The Configure script doesn't recognize your system environment.**
This might be either because it's completely unknown or because the specific environment (include files, OS version, *et cetera*) isn't explicitly handled. If this happens, you may need to port the server to your OS yourself.
- **Your system's C compiler is garbage.**
Some operating systems include a default C compiler that is either not ANSI C-compliant or suffers from other deficiencies. The usual recommendation in cases like this is to acquire, install, and use gcc.
- **Your include files may be confused.**
In some cases, we have found that a compiler installation or system upgrade has left the C header files in an inconsistent state. Make sure that your include directory tree is in sync with the compiler and the operating system.
- **Your operating system or compiler may be out of revision.**
Software vendors (including those that develop operating systems) issue new releases for a reason; sometimes to add functionality, but more often to fix bugs that have been discovered. Try upgrading your compiler and/or your operating system.

The Apache Group tests the ability to build the server on many different platforms. Unfortunately, we can't test all of the OS platforms there are. If you have verified that none of the above issues is the cause of your problem, and it hasn't been reported before, please submit a [problem report](#). Be sure to include *complete* details, such as the compiler & OS versions and exact error messages.

3. Why do I get complaints about redefinition of "struct iovec" when compiling under Linux?

This is a conflict between your C library includes and your kernel includes. You need to make sure that the versions of both are matched properly. There are two workarounds, either one will solve the problem:

- Remove the definition of `struct iovec` from your C library includes. It is located in `/usr/include/sys/uio.h`. **Or,**
 - Add `-DNO_WRITEV` to the `EXTRA_CFLAGS` line in your Configuration and reconfigure/rebuild. This hurts performance and should only be used as a last resort.
-

4. I'm using gcc and I get some compilation errors, what is wrong?

GCC parses your system header files and produces a modified subset which it uses for compiling. This behavior ties GCC tightly to the version of your operating system. So, for example, if you were running IRIX 5.3 when you built GCC and then upgrade to IRIX 6.2 later, you will have to rebuild GCC. Similarly for Solaris 2.4, 2.5, or 2.5.1 when you upgrade to 2.6. Sometimes you can type "gcc -v" and it will tell you the version of the operating system it was built against.

If you fail to do this, then it is very likely that Apache will fail to build. One of the most common errors is with `readv`, `writev`, or `uio.h`. This is **not** a bug with Apache. You will need to re-install GCC.

5. I'm using RedHat Linux 5.0, or some other glibc-based Linux system, and I get errors with the crypt function when I attempt to build Apache 1.2.

glibc puts the `crypt` function into a separate library. Edit your `src/Configuration` file and set this:

```
EXTRA_LIBS=-lcrypt
```

Then re-run `src/Configure` and re-execute the `make`.

D. Error Log Messages and Problems Starting Apache

1. Why do I get "setgid: Invalid argument" at startup?

Your [Group](#) directive (probably in `conf/httpd.conf`) needs to name a group that actually exists in the `/etc/group` file (or your system's equivalent). This problem is also frequently seen when a negative number is used in the `Group` directive (e.g., `Group #-1`). Using a group name -- not group number -- found in your system's group database should solve this problem in all cases.

2. Why am I getting "httpd: could not set socket option TCP_NODELAY" in my error log?

This message almost always indicates that the client disconnected before Apache reached the point of calling `setsockopt()` for the connection. It shouldn't occur for more than about 1% of the requests your server handles, and it's advisory only in any case.

3. Why am I getting "connection reset by peer" in my error log?

This is a normal message and nothing about which to be alarmed. It simply means that the client canceled the connection before it had been completely set up - such as by the end-user pressing the "Stop" button. People's patience being what it is, sites with response-time problems or slow network links may experience this more than high-capacity ones or those with large pipes to the network.

4. The errorlog says Apache dumped core, but where's the dump file?

In Apache version 1.2, the error log message about dumped core includes the directory where the dump file should be located. However, many Unixes do not allow a process that has called `setuid()` to dump core for security reasons; the typical Apache setup has the server started as root to bind to port 80, after which it changes UIDs to a non-privileged user to serve requests.

Dealing with this is extremely operating system-specific, and may require rebuilding your system kernel. Consult your operating system documentation or vendor for more information about whether your system does this and how to bypass it. If there is a documented way of bypassing it, it is recommended that you bypass it only for the httpd server process if possible.

The canonical location for Apache's core-dump files is the [ServerRoot](#) directory. As of Apache version 1.3, the location can be set via the [CoreDumpDirectory](#) directive to a different directory. Make sure that this directory is writable by the user the server runs as (as opposed to the user the server is *started* as).

5. When I run it under Linux I get "shmget: function not found", what should I do?

Your kernel has been built without SysV IPC support. You will have to rebuild the kernel with that support enabled (it's under the "General Setup" submenu). Documentation for kernel building is beyond the scope of this FAQ; you should consult the [Kernel HOWTO](#), or the documentation provided with your distribution, or a [Linux newsgroup/ mailing list](#). As a last-resort workaround, you can comment out the `#define USE_SHMGET_SCOREBOARD` definition in the `LINUX` section of `src/conf.h` and rebuild the server (prior to 1.3b4, simply removing `#define HAVE_SHMGET` would have sufficed). This will produce a server which is slower and less reliable.

6. Server hangs, or fails to start, and/or error log fills with "fcntl: F_SETLKW: No record locks available" or similar messages

These are symptoms of a file locking problem, which usually means that the server is trying to use a synchronization file on an NFS filesystem.

Because of its parallel-operation model, the Apache Web server needs to provide some form of synchronization when accessing certain resources. One of these synchronization methods involves taking out locks on a file, which means that the filesystem whereon the lockfile resides must support locking. In many cases this means it *can't* be kept on an NFS-mounted filesystem.

To cause the Web server to work around the NFS locking limitations, include a line such as the following in your server configuration files:

```
LockFile /var/run/apache-lock
```

The directory should not be generally writable (e.g., don't use `/var/tmp`). See the [LockFile](#) documentation for more information.

7. Why am I getting "Expected </Directory> but saw </Directory>" when I try to start Apache?

This is a known problem with certain versions of the AIX C compiler. IBM are working on a solution, and the issue is being tracked by [problem report #2312](#).

8. I'm using RedHat Linux and I have problems with httpd dying randomly or not restarting properly

RedHat Linux versions 4.x (and possibly earlier) RPMs contain various nasty scripts which do not stop or restart Apache properly. These can affect you even if you're not running the RedHat supplied RPMs.

If you're using the default install then you're probably running Apache 1.1.3, which is outdated. From RedHat's ftp site you can pick up a more recent RPM for Apache 1.2.x. This will solve one of the problems.

If you're using a custom built Apache rather than the RedHat RPMs then you should `rpm -e apache`. In particular you want the mildly broken `/etc/logrotate.d/apache` script to be removed, and you want the broken `/etc/rc.d/init.d/httpd` (or `httpd.init`) script to be removed. The latter is actually fixed by the `apache-1.2.5` RPMs but if you're building your own Apache then you probably don't want the RedHat files.

We can't stress enough how important it is for folks, *especially vendors* to follow the [stopping Apache directions](#) given in our documentation. In RedHat's defense, the broken scripts were necessary with Apache 1.1.x because the Linux support in 1.1.x was very poor, and there were various race conditions on all platforms. None of this should be necessary with Apache 1.2 and later.

9. I upgraded from an Apache version earlier than 1.2.0 and suddenly I have problems with Apache dying randomly or not restarting properly

You should read [the previous note](#) about problems with RedHat installations. It is entirely likely that your installation has start/stop/restart scripts which were built for an earlier version of Apache. Versions earlier than 1.2.0 had various race conditions that made it necessary to use `kill -9` at times to take out all the httpd servers. But that should not be necessary any longer. You should follow the [directions on how to stop and restart Apache](#).

As of Apache 1.3 there is a script `src/support/apachectl` which, after a bit of customization, is suitable for starting, stopping, and restarting your server.

10. When I try to start Apache from a DOS window, I get a message like "Cannot determine host name. Use ServerName directive to set it manually." What does this mean?

It means what it says; the Apache software can't determine the hostname of your system. Edit your `conf\httpd.conf` file, look for the string "ServerName", and make sure there's an uncommented directive such as

```
ServerName localhost
```

or

```
ServerName www.foo.com
```

in the file. Correct it if there one there with wrong information, or add one if you don't already have one.

Also, make sure that your Windows system has DNS enabled. See the TCP/IP setup component of the Networking or Internet Options control panel.

After verifying that DNS is enabled and that you have a valid hostname in your ServerName directive, try to start the server again.

11. When I try to start Apache for Windows, I get a message like "Unable To Locate WS2_32.DLL...". What should I do?

Short answer: You need to install Winsock 2, available from <http://www.microsoft.com/windows95/downloads/>

Detailed answer: Prior to version 1.3.9, Apache for Windows used Winsock 1.1. Beginning with version 1.3.9, Apache began using Winsock 2 features (specifically, `WSADuplicateSocket()`). `WS2_32.DLL` implements the Winsock 2 API. Winsock 2 ships with Windows NT 4.0 and Windows 98. Some of the earlier releases of Windows 95 did not include Winsock 2.

12. Apache for Windows does not start. Error log contains this message: "[crit] (10045) The attempted operation is not supported for the type of object referenced: Parent: WSADuplicateSocket failed for socket ####". What does this mean?

We have seen this problem when Apache is run on systems along with Virtual Private Networking clients like Aventail Connect. Aventail Connect is a Layered Service Provider (LSP) that inserts itself, as a "shim," between the Winsock 2 API and Window's native Winsock 2 implementation. The Aventail Connect shim does not implement WSADuplicateSocket, which is the cause of the failure.

The shim is not unloaded when Aventail Connect is shut down. Once observed, the problem persists until the shim is either explicitly unloaded or the machine is rebooted. Instructions for temporarily removing the Aventail Connect V3.x shim can be found here: "[How to Remove Aventail Connect v3.x from the LSP Order for Testing Purposes.](#)"

Another potential solution (not tested) is to add `apache.exe` to the Aventail "Connect Exclusion List". See this link for details: "[How to Add an Application to Aventail Connect's Application Exclusion List.](#)"

Apache is affected in a similar way by *any* firewall program that isn't correctly configured. Assure you exclude your Apache server ports (usually port 80) from the list of ports to block. Refer to your firewall program's documentation for the how-to.

13. When I try to start Apache on Windows, I get a message like "System error 1067 has occurred. The process terminated unexpectedly." What does this mean?

This message means that the Web server was unable to start correctly for one reason or another. To find out why, execute the following commands in a DOS window:

```
c:
cd "\Program Files\Apache Group\Apache"
apache
```

(If you don't get the prompt back, hit Control-C to cause Apache to exit.)

The error you see will probably be one of those preceding this question in the FAQ.

As of Apache 1.3.14, first check the Windows NT Event Log for Application errors using the Windows NT/2000 Event Viewer program. Any errors that occur prior to opening the Apache error log will be stored here, if Apache is run as a Service on NT or 2000. As with any error, also check your Apache error log.

14. On a SuSE Linux system, I try and configure access control using basic authentication. Although I follow the example exactly, authentication fails, and an error message "admin: not a valid FDN:" is logged.

In the SuSE distribution, additional 3rd party authentication modules have been added and activated by default. These modules interfere with the Apache standard modules and cause Basic authentication to fail. Our recommendation is to comment all those modules in `/etc/httpd/suse_addmodule.conf` and `/etc/httpd/suse_loadmodule.conf` which are not actually required for running your server.

E. Configuration Questions

1. Why can't I run more than <n> virtual hosts?

You are probably running into resource limitations in your operating system. The most common limitation is the *per*-process limit on **file descriptors**, which is almost always the cause of problems seen when adding virtual hosts. Apache often does not give an intuitive error message because it is normally some library routine (such as `gethostbyname()`) which needs file descriptors and doesn't complain intelligibly when it can't get them.

Each log file requires a file descriptor, which means that if you are using separate access and error logs for each virtual host, each virtual host needs two file descriptors. Each [Listen](#) directive also needs a file descriptor.

Typical values for <n> that we've seen are in the neighborhood of 128 or 250. When the server bumps into the file descriptor limit, it may dump core with a SIGSEGV, it might just hang, or it may limp along and you'll see (possibly meaningful) errors in the error log. One common problem that occurs when you run into a file descriptor limit is that CGI scripts stop being executed properly.

As to what you can do about this:

1. Reduce the number of [Listen](#) directives. If there are no other servers running on the machine on the same port then you normally don't need any Listen directives at all. By default Apache listens to all addresses on port 80.

2. Reduce the number of log files. You can use [mod_log_config](#) to log all requests to a single log file while including the name of the virtual host in the log file. You can then write a script to split the logfile into separate files later if necessary. Such a script is provided with the Apache 1.3 distribution in the `src/support/split-logfile` file.
3. Increase the number of file descriptors available to the server (see your system's documentation on the `limit` or `ulimit` commands). For some systems, information on how to do this is available in the [performance hints](#) page. There is a specific note for [FreeBSD](#) below.

For Windows 95, try modifying your `C:\CONFIG.SYS` file to include a line like

```
FILES=300
```

Remember that you'll need to reboot your Windows 95 system in order for the new value to take effect.

4. "Don't do that" - try to run with fewer virtual hosts
5. Spread your operation across multiple server processes (using [Listen](#) for example, but see the first point) and/or ports.

Since this is an operating-system limitation, there's not much else available in the way of solutions.

As of 1.2.1 we have made attempts to work around various limitations involving running with many descriptors. [More information is available.](#)

2. Can I increase `FD_SETSIZE` on FreeBSD?

On versions of FreeBSD before 3.0, the `FD_SETSIZE` define defaults to 256. This means that you will have trouble usefully using more than 256 file descriptors in Apache. This can be increased, but doing so can be tricky.

If you are using a version prior to 2.2, you need to recompile your kernel with a larger `FD_SETSIZE`. This can be done by adding a line such as:

```
options FD_SETSIZE nnn
```

to your kernel config file. Starting at version 2.2, this is no longer necessary.

If you are using a version of 2.1-stable from after 1997/03/10 or 2.2 or 3.0-current from before 1997/06/28, there is a limit in the resolver library that prevents it from using more file descriptors than what `FD_SETSIZE` is set to when `libc` is compiled. To increase this, you have to recompile `libc` with a higher `FD_SETSIZE`.

In FreeBSD 3.0, the default `FD_SETSIZE` has been increased to 1024 and the above limitation in the resolver library has been removed.

After you deal with the appropriate changes above, you can increase the setting of `FD_SETSIZE` at Apache compilation time by adding `"-DFD_SETSIZE=nnn"` to the `EXTRA_CFLAGS` line in your Configuration file.

3. Why doesn't my `ErrorDocument 401` work?

You need to use it with a URL in the form `"/foo/bar"` and not one with a method and hostname such as `"http://host/foo/bar"`. See the [ErrorDocument](#) documentation for details. This was incorrectly documented in the past.

4. Why does Apache send a cookie on every response?

Apache does *not* automatically send a cookie on every response, unless you have re-compiled it with the [mod_usertrack](#) module, and specifically enabled it with the [CookieTracking](#) directive. This module has been in Apache since version 1.2. This module may help track users, and uses cookies to do this. If you are not using the data generated by `mod_usertrack`, do not compile it into Apache.

5. Why don't my cookies work, I even compiled in `mod_cookies`?

Firstly, you do *not* need to compile in `mod_cookies` in order for your scripts to work (see the [previous question](#) for more about `mod_cookies`). Apache passes on your `Set-Cookie` header fine, with or without this module. If cookies do not work it will be because your script does not work properly or your browser does not use cookies or is not set-up to accept them.

6. Why do my Java app[let]s give me plain text when I request an URL from an Apache server?

As of version 1.2, Apache is an HTTP/1.1 (HyperText Transfer Protocol version 1.1) server. This fact is reflected in the protocol version that's included in the response headers sent to a client when processing a request. Unfortunately, low-level Web access classes included in the Java Development Kit (JDK) version 1.0.2 expect to see the version string "HTTP/1.0" and do not correctly interpret the "HTTP/1.1" value Apache is sending (this part of the response is a declaration of what the server can do rather than a declaration of the dialect of the response). The result is that the JDK methods do not correctly parse the headers, and include them with the document content by mistake.

This is definitely a bug in the JDK 1.0.2 foundation classes from Sun, and it has been fixed in version 1.1. However, the classes in question are part of the virtual machine environment, which means they're part of the Web browser (if Java-enabled) or the Java environment on the client system - so even if you develop *your* classes with a recent JDK, the eventual users might encounter the problem. The classes involved are replaceable by vendors implementing the Java virtual machine environment, and so even those that are based upon the 1.0.2 version may not have this problem.

In the meantime, a workaround is to tell Apache to "fake" an HTTP/1.0 response to requests that come from the JDK methods; this can be done by including a line such as the following in your server configuration files:

```
BrowserMatch Java1.0 force-response-1.0
BrowserMatch JDK/1.0 force-response-1.0
```

More information about this issue can be found in the [Java and HTTP/1.1](#) page at the Apache web site.

7. How do I get Apache to send a MIDI file so the browser can play it?

Even though the registered MIME type for MIDI files is audio/midi, some browsers are not set up to recognize it as such; instead, they look for audio/x-midi. There are two things you can do to address this:

1. Configure your browser to treat documents of type audio/midi correctly. This is the type that Apache sends by default. This may not be workable, however, if you have many client installations to change, or if some or many of the clients are not under your control.
2. Instruct Apache to send a different Content-type header for these files by adding the following line to your server's configuration files:

```
AddType audio/x-midi .mid .midi .kar
```

Note that this may break browsers that *do* recognize the audio/midi MIME type unless they're prepared to also handle audio/x-midi the same way.

8. How do I add browsers and referrers to my logs?

Apache provides a couple of different ways of doing this. The recommended method is to compile the [mod_log_config](#) module into your configuration and use the [CustomLog](#) directive.

You can either log the additional information in files other than your normal transfer log, or you can add them to the records already being written. For example:

```
CustomLog logs/access_log "%h %l %u %t \"%r\" %s %b \"%{Referer}i\" \"%{User-Agent}i\""
```

This will add the values of the User-agent: and Referer: headers, which indicate the client and the referring page, respectively, to the end of each line in the access log.

You may want to check out the Apache Week article entitled: "[Gathering Visitor Information: Customizing Your Logfiles](#)".

9. Why does accessing directories only work when I include the trailing "/" (e.g., http://foo.domain.com/~user/) but not when I omit it (e.g., http://foo.domain.com/~user)?

When you access a directory without a trailing "/", Apache needs to send what is called a redirect to the client to tell it to add the trailing slash. If it did not do so, relative URLs would not work properly. When it sends the redirect, it needs to know the name of the server so that it can include it in the redirect. There are two ways for Apache to find this out; either it can guess, or you can tell it. If your DNS is configured correctly, it can normally guess without any problems. If it is not, however, then you need to tell it.

Add a [ServerName](#) directive to the config file to tell it what the domain name of the server is.

10. Why doesn't mod_info list any directives?

The [mod_info](#) module allows you to use a Web browser to see how your server is configured. Among the information it displays is the list modules and their configuration directives. The "current" values for the directives are not necessarily those of the running server; they are extracted from the configuration files themselves at the time of the request. If the files have been changed since the server was last reloaded, the display will not match the values actively in use. If the files and the path to the files are not readable by the user as which the server is running (see the [User](#) directive), then mod_info cannot read them in order to list their values. An entry *will* be made in the error log in this event, however.

11. I upgraded to Apache 1.3 and now my virtual hosts don't work!

In versions of Apache prior to 1.3b2, there was a lot of confusion regarding address-based virtual hosts and (HTTP/1.1) name-based virtual hosts, and the rules concerning how the server processed <VirtualHost> definitions were very complex and not well documented.

Apache 1.3b2 introduced a new directive, [NameVirtualHost](#), which simplifies the rules quite a bit. However, changing the rules like this means that your existing name-based <VirtualHost> containers probably won't work correctly immediately following the upgrade.

To correct this problem, add the following line to the beginning of your server configuration file, before defining any virtual hosts:

```
NameVirtualHost n.n.n.n
```

Replace the "n.n.n.n" with the IP address to which the name-based virtual host names resolve; if you have multiple name-based hosts on multiple addresses, repeat the directive for each address.

Make sure that your name-based <VirtualHost> blocks contain ServerName and possibly ServerAlias directives so Apache can be sure to tell them apart correctly.

Please see the [Apache Virtual Host documentation](#) for further details about configuration.

12. I'm using RedHat Linux and my .htm files are showing up as HTML source rather than being formatted!

RedHat messed up and forgot to put a content type for .htm files into /etc/mime.types. Edit /etc/mime.types, find the line containing html and add htm to it. Then restart your httpd server:

```
kill -HUP `cat /var/run/httpd.pid`
```

Then **clear your browsers' caches**. (Many browsers won't re-examine the content type after they've reloaded a page.)

13. My .htaccess files are being ignored.

This is almost always due to your [AllowOverride](#) directive being set incorrectly for the directory in question. If it is set to None then .htaccess files will not even be looked for. If you do have one that is set, then be certain it covers the directory you are trying to use the .htaccess file in. This is normally accomplished by ensuring it is inside the proper [Directory](#) container.

14. Why do I get a "Forbidden" message whenever I try to access a particular directory?

This message is generally caused because either

- The underlying file system permissions do not allow the User/Group under which Apache is running to access the necessary files; or
- The Apache configuration has some access restrictions in place which forbid access to the files.

You can determine which case applies to your situation by checking the error log.

In the case where file system permission are at fault, remember that not only must the directory and files in question be readable, but also all parent directories must be at least searchable by the web server in order for the content to be accessible.

15. Why do I get a "Forbidden/You don't have permission to access / on this server" message whenever I try to access my server?

Search your `conf/httpd.conf` file for this exact string: `<Files ~>`. If you find it, that's your problem -- that particular `<Files>` container is malformed. Delete it or replace it with `<Files ~ "^\.ht">` and restart your server and things should work as expected.

This error appears to be caused by a problem with the version of `linuxconf` distributed with Redhat 6.x. It may reappear if you use `linuxconf` again.

If you don't find this string, check out the [previous question](#).

16. Why do my files appear correctly in Internet Explorer, but show up as source or trigger a save window with Netscape?

Internet Explorer (IE) and Netscape handle mime type detection in different ways, and therefore will display the document differently. In particular, IE sometimes relies on the file extension to determine the mime type. This can happen when the server specifies a mime type of `application/octet-stream` or `text/plain`. (Unfortunately, this behavior makes it impossible to properly send plain text in some situations unless the file extension is `txt`.) There are more details available on IE's mime type detection behavior in an [MSDN article](#).

In order to make all browsers work correctly, you should assure that Apache sends the correct mime type for the file. This is accomplished by editing the `mime.types` file or using an [AddType](#) directive in the Apache configuration files.

17. My site is accessible under many different hostnames; how do I redirect clients so that they see only a single name?

Many sites map a variety of hostnames to the same content. For example, `www.example.com`, `example.com` and `www.example.net` may all refer to the same site. It is best to make sure that, regardless of the name clients use to access the site, they will be redirected to a single, canonical hostname. This makes the site easier to maintain and assures that there will be only one version of the site in proxy caches and search engines.

There are two techniques to implement canonical hostnames:

1. Use [mod_rewrite](#) as described in the "Canonical Hostnames" section of the [URL Rewriting Guide](#).
2. Use [name-based virtual hosting](#):

```
NameVirtualHost *

<VirtualHost *>
    ServerName www.example.net
    ServerAlias example.com
    Redirect permanent / http://www.example.com/
</VirtualHost>

<VirtualHost *>
    ServerName www.example.com
    DocumentRoot /usr/local/apache/htdocs
</VirtualHost>
```

F. Dynamic Content (CGI and SSI)

1. How do I enable CGI execution in directories other than the ScriptAlias?

Apache recognizes all files in a directory named as a [ScriptAlias](#) as being eligible for execution rather than processing as normal documents. This applies regardless of the file name, so scripts in a `ScriptAlias` directory don't need to be named `*.cgi` or `*.pl` or whatever. In other words, *all* files in a `ScriptAlias` directory are scripts, as far as Apache is concerned.

To persuade Apache to execute scripts in other locations, such as in directories where normal documents may also live, you must tell it how to recognize them - and also that it's okay to execute them. For this, you need to use something like the [AddHandler](#) directive.

1. In an appropriate section of your server configuration files, add a line such as

```
AddHandler cgi-script .cgi
```

The server will then recognize that all files in that location (and its logical descendants) that end in `.cgi` are script files, not documents.

2. Make sure that the directory location is covered by an [Options](#) declaration that includes the ExecCGI option.

In some situations, you might not want to actually allow all files named "*.cgi" to be executable. Perhaps all you want is to enable a particular file in a normal directory to be executable. This can be alternatively accomplished *via* [mod_rewrite](#) and the following steps:

1. Locally add to the corresponding .htaccess file a ruleset similar to this one:

```
RewriteEngine on
RewriteBase /~foo/bar/
RewriteRule ^quux\.cgi$ - [T=application/x-httpd-cgi]
```

2. Make sure that the directory location is covered by an [Options](#) declaration that includes the ExecCGI and FollowSymLinks option.

2. What does it mean when my CGIs fail with "Premature end of script headers"?

It means just what it says: the server was expecting a complete set of HTTP headers (one or more followed by a blank line), and didn't get them.

The most common cause of this problem is the script dying before sending the complete set of headers, or possibly any at all, to the server. To see if this is the case, try running the script standalone from an interactive session, rather than as a script under the server. If you get error messages, this is almost certainly the cause of the "premature end of script headers" message. Even if the CGI runs fine from the command line, remember that the environment and permissions may be different when running under the web server. The CGI can only access resources allowed for the [User](#) and [Group](#) specified in your Apache configuration. In addition, the environment will not be the same as the one provided on the command line, but it can be adjusted using the directives provided by [mod_env](#).

The second most common cause of this (aside from people not outputting the required headers at all) is a result of an interaction with Perl's output buffering. To make Perl flush its buffers after each output statement, insert the following statements around the `print` or `write` statements that send your HTTP headers:

```
{
  local ($oldbar) = $|;
  $cfh = select (STDOUT);
  $| = 1;
  #
  # print your HTTP headers here
  #
  $| = $oldbar;
  select ($cfh);
}
```

This is generally only necessary when you are calling external programs from your script that send output to stdout, or if there will be a long delay between the time the headers are sent and the actual content starts being emitted. To maximize performance, you should turn buffer-flushing back *off* (with `$| = 0` or the equivalent) after the statements that send the headers, as displayed above.

If your script isn't written in Perl, do the equivalent thing for whatever language you *are* using (*e.g.*, for C, call `fflush()` after writing the headers).

Another cause for the "premature end of script headers" message are the `RLimitCPU` and `RLimitMEM` directives. You may get the message if the CGI script was killed due to a resource limit.

In addition, a configuration problem in [suEXEC](#), `mod_perl`, or another third party module can often interfere with the execution of your CGI and cause the "premature end of script headers" message.

3. Why do I keep getting "Method Not Allowed" for form POST requests?

This is almost always due to Apache not being configured to treat the file you are trying to POST to as a CGI script. You can not POST to a normal HTML file; the operation has no meaning. See the FAQ entry on [CGIs outside ScriptAliased directories](#) for details on how to configure Apache to treat the file in question as a CGI.

4. How can I get my script's output without Apache buffering it? Why doesn't my server push work?

As of Apache 1.3, CGI scripts are essentially not buffered. Every time your script does a "flush" to output data, that data gets relayed on to the client. Some scripting languages, for example Perl, have their own buffering for output - this can be disabled by setting the `$|` special variable to 1. Of course this does increase the overall number of packets being transmitted, which can result in a sense of slowness for the end user.

Prior to 1.3, you needed to use "nph-" scripts to accomplish non-buffering. Today, the only difference between nph scripts and normal scripts is that nph scripts require the full HTTP headers to be sent.

5. Where can I find the "CGI specification"?

The Common Gateway Interface (CGI) specification can be found at the original NCSA site <<http://hoohoo.ncsa.uiuc.edu/cgi/interface.html>>. This version hasn't been updated since 1995, and there have been some efforts to update it.

A new draft is being worked on with the intent of making it an informational RFC; you can find out more about this project at <<http://web.golux.com/coar/cgi/>>.

6. Why isn't FastCGI included with Apache any more?

The simple answer is that it was becoming too difficult to keep the version being included with Apache synchronized with the master copy at the [FastCGI web site](#). When a new version of Apache was released, the version of the FastCGI module included with it would soon be out of date.

You can still obtain the FastCGI module for Apache from the master FastCGI web site.

7. How do I enable SSI (parsed HTML)?

SSI (an acronym for Server-Side Include) directives allow static HTML documents to be enhanced at run-time (*e.g.*, when delivered to a client by Apache). The format of SSI directives is covered in the [mod_include manual](#); suffice it to say that Apache supports not only SSI but xSSI (eXtended SSI) directives.

Processing a document at run-time is called *parsing* it; hence the term "parsed HTML" sometimes used for documents that contain SSI instructions. Parsing tends to be resource-consumptive compared to serving static files, and is not enabled by default. It can also interfere with the cachability of your documents, which can put a further load on your server. (See the [next question](#) for more information about this.)

To enable SSI processing, you need to

- Build your server with the [mod_include](#) module. This is normally compiled in by default.
- Make sure your server configuration files have an [Options](#) directive which permits Includes.
- Make sure that the directory where you want the SSI documents to live is covered by the "server-parsed" content handler, either explicitly or in some ancestral location. That can be done with the following [AddHandler](#) directive:

```
AddHandler server-parsed .shtml
```

This indicates that all files ending in ".shtml" in that location (or its descendants) should be parsed. Note that using ".html" will cause all normal HTML files to be parsed, which may put an inordinate load on your server.

For additional information, see the Apache Week article on [Using Server Side Includes](#).

8. Why don't my parsed files get cached?

Since the server is performing run-time processing of your SSI directives, which may change the content shipped to the client, it can't know at the time it starts parsing what the final size of the result will be, or whether the parsed result will always be the same. This means that it can't generate Content-Length or Last-Modified headers. Caches commonly work by comparing the Last-Modified of what's in the cache with that being delivered by the server. Since the server isn't sending that header for a parsed document, whatever's doing the caching can't tell whether the document has changed or not - and so fetches it again to be on the safe side.

You can work around this in some cases by causing an Expires header to be generated. (See the [mod_expires](#) documentation for more details.) Another possibility is to use the [XBitHack Full](#) mechanism, which tells Apache to send (under certain circumstances detailed in the XBitHack directive description) a Last-Modified header based upon the last modification time of the

file being parsed. Note that this may actually be lying to the client if the parsed file doesn't change but the SSI-inserted content does; if the included content changes often, this can result in stale copies being cached.

9. How can I have my script output parsed?

So you want to include SSI directives in the output from your CGI script, but can't figure out how to do it? The short answer is "you can't." This is potentially a security liability and, more importantly, it can not be cleanly implemented under the current server API. The best workaround is for your script itself to do what the SSIs would be doing. After all, it's generating the rest of the content.

This is a feature The Apache Group hopes to add in the next major release after 1.3.

10. SSIs don't work for VirtualHosts and/or user home directories.

This is almost always due to having some setting in your config file that sets "Options Includes" or some other setting for your DocumentRoot but not for other directories. If you set it inside a Directory section, then that setting will only apply to that directory.

11. How can I use ErrorDocument and SSI to simplify customized error messages?

Have a look at [this document](#). It shows in example form how you can a combination of XSSSI and negotiation to tailor a set of ErrorDocuments to your personal taste, and returning different internationalized error responses based on the client's native language.

12. Why is the environment variable REMOTE_USER not set?

This variable is set and thus available in SSI or CGI scripts **if and only if** the requested document was protected by access authentication. For an explanation on how to implement these restrictions, see [Apache Week's](#) articles on [Using User Authentication](#) or [DBM User Authentication](#).

Hint: When using a CGI script to receive the data of a HTML FORM notice that protecting the document containing the FORM is not sufficient to provide REMOTE_USER to the CGI script. You have to protect the CGI script, too. Or alternatively only the CGI script (then authentication happens only after filling out the form).

13. How do I allow each of my user directories to have a cgi-bin directory?

Remember that CGI execution does not need to be restricted only to cgi-bin directories. You can [allow CGI script execution in arbitrary parts of your filesystem](#).

There are many ways to give each user directory a cgi-bin directory such that anything requested as `http://example.com/~user/cgi-bin/program` will be executed as a CGI script. Two alternatives are:

1. Place the cgi-bin directory next to the public_html directory:

```
ScriptAliasMatch ^/~([^/]+)/cgi-bin/(.*) /home/$1/cgi-bin/$2
```

2. Place the cgi-bin directory underneath the public_html directory:

```
<Directory /home/*/public_html/cgi-bin>
    Options ExecCGI
    SetHandler cgi-script
</Directory>
```

G. Authentication and Access Restrictions

1. Why isn't restricting access by host or domain name working correctly?

Two of the most common causes of this are:

1. **An error, inconsistency, or unexpected mapping in the DNS registration**

This happens frequently: your configuration restricts access to Host.FooBar.Com, but you can't get in from that host. The usual reason for this is that Host.FooBar.Com is actually an alias for another name, and when Apache performs the address-to-name lookup it's getting the *real* name, not Host.FooBar.Com. You can verify this by checking the reverse

lookup yourself. The easiest way to work around it is to specify the correct host name in your configuration.

2. Inadequate checking and verification in your configuration of Apache

If you intend to perform access checking and restriction based upon the client's host or domain name, you really need to configure Apache to double-check the origin information it's supplied. You do this by adding the `-DMAXIMUM_DNS` clause to the `EXTRA_CFLAGS` definition in your Configuration file. For example:

```
EXTRA_CFLAGS=-DMAXIMUM_DNS
```

This will cause Apache to be very paranoid about making sure a particular host address is *really* assigned to the name it claims to be. Note that this *can* incur a significant performance penalty, however, because of all the name resolution requests being sent to a nameserver.

2. How do I set up Apache to require a username and password to access certain documents?

There are several ways to do this; some of the more popular ones are to use the [mod_auth](#), [mod_auth_db](#), or [mod_auth_dbm](#) modules.

For an explanation on how to implement these restrictions, see [Apache Week's](#) articles on [Using User Authentication](#) or [DBM User Authentication](#).

3. How do I set up Apache to allow access to certain documents only if a site is either a local site *or* the user supplies a password and username?

Use the [Satisfy](#) directive, in particular the `Satisfy Any` directive, to require that only one of the access restrictions be met. For example, adding the following configuration to a `.htaccess` or server configuration file would restrict access to people who either are accessing the site from a host under `domain.com` or who can supply a valid username and password:

```
Deny from all
Allow from .domain.com
AuthType Basic
AuthUserFile /usr/local/apache/conf/htpasswd.users
AuthName "special directory"
Require valid-user
Satisfy any
```

See the [user authentication](#) question and the [mod_access](#) module for details on how the above directives work.

4. Why does my authentication give me a server error?

Under normal circumstances, the Apache access control modules will pass unrecognized user IDs on to the next access control module in line. Only if the user ID is recognized and the password is validated (or not) will it give the usual success or "authentication failed" messages.

However, if the last access module in line 'declines' the validation request (because it has never heard of the user ID or because it is not configured), the `http_request` handler will give one of the following, confusing, errors:

- check access
- check user. No user file?
- check access. No groups file?

This does *not* mean that you have to add an `'AuthUserFile /dev/null'` line as some magazines suggest!

The solution is to ensure that at least the last module is authoritative and **CONFIGURED**. By default, `mod_auth` is authoritative and will give an OK/Denied, but only if it is configured with the proper `AuthUserFile`. Likewise, if a valid group is required. (Remember that the modules are processed in the reverse order from that in which they appear in your compile-time Configuration file.)

A typical situation for this error is when you are using the `mod_auth_dbm`, `mod_auth_mysql`, `mod_auth_mysql`, `mod_auth_anon` or `mod_auth_cookie` modules on their own. These are by default **not** authoritative, and this will pass the buck on to the (non-existent) next authentication module when the user ID is not in their respective database. Just add the appropriate `'XXXAuthoritative yes'` line to the configuration.

In general it is a good idea (though not terribly efficient) to have the file-based `mod_auth` a module of last resort. This allows you

to access the web server with a few special passwords even if the databases are down or corrupted. This does cost a file open/seek/close for each request in a protected area.

5. Do I have to keep the (mSQL) authentication information on the same machine?

Some organizations feel very strongly about keeping the authentication information on a different machine than the webserver. With the `mod_auth_mysql`, `mod_auth_mysql`, and other SQL modules connecting to (R)DBMs this is quite possible. Just configure an explicit host to contact.

Be aware that with mSQL and Oracle, opening and closing these database connections is very expensive and time consuming. You might want to look at the code in the `auth_*` modules and play with the compile time flags to alleviate this somewhat, if your RDBMS licences allow for it.

6. Why is my mSQL authentication terribly slow?

You have probably configured the Host by specifying a FQHN, and thus the libmysql will use a full blown TCP/IP socket to talk to the database, rather than a fast internal device. The libmysql, the mSQL FAQ, and the `mod_auth_mysql` documentation warn you about this. If you have to use different hosts, check out the `mod_auth_mysql` code for some compile time flags which might - or might not - suit you.

7. Can I use my /etc/passwd file for Web page authentication?

Yes, you can - but it's a **very bad idea**. Here are some of the reasons:

- The Web technology provides no governors on how often or how rapidly password (authentication failure) retries can be made. That means that someone can hammer away at your system's root password using the Web, using a dictionary or similar mass attack, just as fast as the wire and your server can handle the requests. Most operating systems these days include attack detection (such as n failed passwords for the same account within m seconds) and evasion (breaking the connection, disabling the account under attack, disabling *all* logins from that source, *et cetera*), but the Web does not.
- An account under attack isn't notified (unless the server is heavily modified); there's no "You have 19483 login failures" message when the legitimate owner logs in.
- Without an exhaustive and error-prone examination of the server logs, you can't tell whether an account has been compromised. Detecting that an attack has occurred, or is in progress, is fairly obvious, though - *if you look at the logs*.
- Web authentication passwords (at least for Basic authentication) generally fly across the wire, and through intermediate proxy systems, in what amounts to plain text. "O'er the net we go/Caching all the way;/O what fun it is to surf/Giving my password away!"
- Since HTTP is stateless, information about the authentication is transmitted *each and every time* a request is made to the server. Essentially, the client caches it after the first successful access, and transmits it without asking for all subsequent requests to the same server.
- It's relatively trivial for someone on your system to put up a page that will steal the cached password from a client's cache without them knowing. Can you say "password grabber"?

If you still want to do this in light of the above disadvantages, the method is left as an exercise for the reader. It'll void your Apache warranty, though, and you'll lose all accumulated UNIX guru points.

8. Why does Apache ask for my password twice before serving a file?

If the hostname under which you are accessing the server is different than the hostname specified in the `ServerName` directive, then depending on the setting of the `UseCanonicalName` directive, Apache will redirect you to a new hostname when constructing self-referential URLs. This happens, for example, in the case where you request a directory without including the trailing slash.

When this happens, Apache will ask for authentication once under the original hostname, perform the redirect, and then ask again under the new hostname. For security reasons, the browser must prompt again for the password when the host name changes.

To eliminate this problem you should

1. Always use the trailing slash when requesting directories;
 2. Change the `ServerName` to match the name you are using in the URL; and/or
 3. Set `UseCanonicalName off`.
-

H. URL Rewriting

1. Where can I find mod_rewrite rulesets which already solve particular URL-related problems?

There is a collection of [Practical Solutions for URL-Manipulation](#) where you can find all typical solutions the author of [mod_rewrite](#) currently knows of. If you have more interesting rulesets which solve particular problems not currently covered in this document, send it to [Ralf S. Engelschall](#) for inclusion. The other webmasters will thank you for avoiding the reinvention of the wheel.

2. Where can I find any published information about URL-manipulations and mod_rewrite?

There is an article from [Ralf S. Engelschall](#) about URL-manipulations based on [mod_rewrite](#) in the "iX Multiuser Multitasking Magazin" issue #12/96. The german (original) version can be read online at <<http://www.heise.de/ix/artikel/9612149/>>, the English (translated) version can be found at <<http://www.heise.de/ix/artikel/E/9612149/>>.

3. Why is mod_rewrite so difficult to learn and seems so complicated?

Hmmm... there are a lot of reasons. First, mod_rewrite itself is a powerful module which can help you in really **all** aspects of URL rewriting, so it can be no trivial module per definition. To accomplish its hard job it uses software leverage and makes use of a powerful regular expression library by Henry Spencer which is an integral part of Apache since its version 1.2. And regular expressions itself can be difficult to newbies, while providing the most flexible power to the advanced hacker.

On the other hand mod_rewrite has to work inside the Apache API environment and needs to do some tricks to fit there. For instance the Apache API as of 1.x really was not designed for URL rewriting at the `.htaccess` level of processing. Or the problem of multiple rewrites in sequence, which is also not handled by the API per design. To provide this features mod_rewrite has to do some special (but API compliant!) handling which leads to difficult processing inside the Apache kernel. While the user usually doesn't see anything of this processing, it can be difficult to find problems when some of your RewriteRules seem not to work.

4. What can I do if my RewriteRules don't work as expected?

Use "RewriteLog somefile" and "RewriteLogLevel 9" and have a precise look at the steps the rewriting engine performs. This is really the only one and best way to debug your rewriting configuration.

5. Why don't some of my URLs get prefixed with DocumentRoot when using mod_rewrite?

If the rule starts with /somedir/... make sure that really no /somedir exists on the filesystem if you don't want to lead the URL to match this directory, *i.e.*, there must be no root directory named somedir on the filesystem. Because if there is such a directory, the URL will not get prefixed with DocumentRoot. This behavior looks ugly, but is really important for some other aspects of URL rewriting.

6. How can I make all my URLs case-insensitive with mod_rewrite?

You can't! The reasons are: first, that, case translations for arbitrary length URLs cannot be done *via* regex patterns and corresponding substitutions. One needs a per-character pattern like the sed/Perl `tr|.|.` feature. Second, just making URLs always upper or lower case does not solve the whole problem of case-**INSENSITIVE** URLs, because URLs actually have to be rewritten to the correct case-variant for the file residing on the filesystem in order to allow Apache to access the file. And the Unix filesystem is always case-**SENSITIVE**.

But there is a module named [mod_speling.c](#) in the Apache distribution. Try this module to help correct people who use mis-cased URLs.

7. Why are RewriteRules in my VirtualHost parts ignored?

Because you have to enable the engine for every virtual host explicitly due to security concerns. Just add a "RewriteEngine on" to your virtual host configuration parts.

8. How can I use strings with whitespaces in RewriteRule's ENV flag?

There is only one ugly solution: You have to surround the complete flag argument by quotation marks ("[E=...]"). Notice: The argument to quote here is not the argument to the E-flag, it is the argument of the Apache config file parser, *i.e.*, the third argument of the RewriteRule here. So you have to write "[E=any text with whitespaces]".

I. Features

1. Does or will Apache act as a Proxy server?

Apache version 1.1 and above comes with a [proxy module](#). If compiled in, this will make Apache act as a caching-proxy server.

2. What are "multiviews"?

"Multiviews" is the general name given to the Apache server's ability to provide language-specific document variants in response to a request. This is documented quite thoroughly in the [content negotiation](#) description page. In addition, Apache Week carried an article on this subject entitled "[Content Negotiation Explained](#)".

3. Why can't I publish to my Apache server using PUT on Netscape Gold and other programs?

Because you need to install and configure a script to handle the uploaded files. This script is often called a "PUT" handler. There are several available, but they may have security problems. Using FTP uploads may be easier and more secure, at least for now. For more information, see the Apache Week article [Publishing Pages with PUT](#).

4. Why doesn't Apache include SSL?

SSL (Secure Socket Layer) data transport requires encryption, and many governments have restrictions upon the import, export, and use of encryption technology. If Apache included SSL in the base package, its distribution would involve all sorts of legal and bureaucratic issues, and it would no longer be freely available. Also, some of the technology required to talk to current clients using SSL is patented by [RSA Data Security](#), who restricts its use without a license.

Some SSL implementations of Apache are available, however; see the "[related projects](#)" page at the main Apache web site.

You can find out more about this topic in the Apache Week article about [Apache and Secure Transactions](#).

5. How can I attach a footer to my documents without using SSI?

You can make arbitrary changes to static documents by configuring an [Action](#) which launches a CGI script. The CGI is then responsible for setting a content-type and delivering the requested document (the location of which is passed in the PATH_TRANSLATED environment variable), along with whatever footer is needed.

Busy sites may not want to run a CGI script on every request, and should consider using an Apache module to add the footer. There are several third party modules available through the [Apache Module Registry](#) which will add footers to documents. These include mod_trailer, PHP (php3_auto_append_file), mod_layout, and mod_perl (Apache::Sandwich).

6. Does Apache include a search engine?

Apache does not include a search engine, but there are many good commercial and free search engines which can be used easily with Apache. Some of them are listed on the [Web Site Search Tools](#) page. Open source search engines that are often used with Apache include [ht://Dig](#) and [SWISH-E](#).

7. How can I rotate my log files?

The simple answer: by piping the transfer log into an appropriate log file rotation utility.

The longer answer: In the src/support/ directory, you will find a utility called [rotatelogs](#) which can be used like this:

```
TransferLog "|/path/to/rotatelogs /path/to/logs/access_log 86400"
```

to enable daily rotation of the log files.

A more sophisticated solution of a logfile rotation utility is available under the name `cronolog` from Andrew Ford's site at <http://www.ford-mason.co.uk/resources/cronolog/>. It can automatically create logfile subdirectories based on time and date, and can have a constant symlink point to the rotating logfiles. (As of version 1.6.1, `cronolog` is available under the [Apache License](#)). Use it like this:

```
CustomLog "|/path/to/cronolog --symlink=/usr/local/apache/logs/access_log  
/usr/local/apache/logs/%Y/%m/access_log" combined
```

8. How do I keep certain requests from appearing in my logs?

The maximum flexibility for removing unwanted information from log files is obtained by post-processing the logs, or using piped-logs to feed the logs through a program which does whatever you want. However, Apache does offer the ability to prevent requests from ever appearing in the log files. You can do this by using the [SetEnvIf](#) directive to set an environment variable for certain requests and then using the conditional [CustomLog](#) syntax to prevent logging when the environment variable is set.

9. Does Apache support any sort of database integration?

No. Apache is a Web (HTTP) server, not an application server. The base package does not include any such functionality. See the [PHP project](#) and the [mod_perl project](#) for examples of modules that allow you to work with databases from within the Apache environment.

10. Can I use Active Server Pages (ASP) with Apache?

The base Apache Web server package does not include ASP support. However, there are a couple of after-market solutions that let you add this functionality; see the [related projects](#) page to find out more.

11. Does Apache come with Java support?

The base Apache Web server package does not include support for Java, Java Server Pages, Enterprise Java Beans, or Java servlets. Those features are available as add-ons from the Apache/Java project site, <URL:<http://jakarta.apache.org/>>.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Authentication, Authorization, and Access Control

- [Introduction](#)
 - [Basic authentication](#)
 - [How basic authentication works](#)
 - [Configuration: Protecting content with basic authentication](#)
 - [Frequently asked questions about basic auth](#)
 - [Security caveat](#)
 - [Digest authentication](#)
 - [How digest auth works](#)
 - [Configuration: Protecting content with digest authentication](#)
 - [Caveats](#)
 - [Database authentication modules](#)
 - [mod_auth_db and mod_auth_dbm](#)
 - [Berkeley DB files](#)
 - [Installing mod_auth_db](#)
 - [Protecting a directory with mod_auth_db](#)
 - [Access control](#)
 - [Allow and Deny](#)
 - [Satisfy](#)
 - [Summary](#)
-

Authentication, Authorization, and Access Control

Introduction

Apache has three distinct ways of dealing with the question of whether a particular request for a resource will result in that resource actually be returned. These criteria are called *Authorization*, *Authentication*, and *Access control*.

Authentication is any process by which you verify that someone is who they claim they are. This usually involves a username and a password, but can include any other method of demonstrating identity, such as a smart card, retina scan, voice recognition, or fingerprints. Authentication is equivalent to showing your drivers license at the ticket counter at the airport.

Authorization is finding out if the person, once identified, is permitted to have the resource. This is usually determined by finding out if that person is a part of a particular group, if that person has paid admission, or has a particular level of security clearance. Authorization is equivalent to checking the guest list at an exclusive party, or checking for your ticket when you go to the opera.

Finally, access control is a much more general way of talking about controlling access to a web resource. Access can be granted or denied based on a wide variety of criteria, such as the network address of the client, the time of day, the phase of the moon, or the browser which the visitor is using. Access control is analogous to locking the gate at closing time, or only letting people onto the ride who are more than 48 inches tall - it's controlling entrance by some arbitrary condition which may or may not have anything to do with the attributes of the particular visitor.

Because these three techniques are so closely related in most real applications, it is difficult to talk about them separate from one another. In particular, authentication and authorization are, in most actual implementations, inextricable.

If you have information on your web site that is sensitive, or intended for only a small group of people, the techniques in this tutorial will help you make sure that the people that see those pages are the people that you wanted to see them.

Basic authentication

As the name implies, basic authentication is the simplest method of authentication, and for a long time was the most common authentication method used. However, other methods of authentication have recently passed basic in common usage, due to usability issues that will be discussed in a minute.

How basic authentication works

When a particular resource has been protected using basic authentication, Apache sends a `401 Authentication Required` header with the response to the request, in order to notify the client that user credentials must be supplied in order for the resource to be returned as requested.

Upon receiving a `401` response header, the client's browser, if it supports basic authentication, will ask the user to supply a username and password to be sent to the server. If you are using a graphical browser, such as Netscape or Internet Explorer, what you will see is a box which pops up and gives you a place to type in your username and password, to be sent back to the server. If the username is in the approved list, and if the password supplied is correct, the resource will be returned to the client.

Because the HTTP protocol is stateless, each request will be treated in the same way, even though they are from the same client. That is, every resource which is requested from the server will have to supply authentication credentials over again in order to receive the resource.

Fortunately, the browser takes care of the details here, so that you only have to type in your username and password one time per browser session - that is, you might have to type it in again the next time you open up your browser and visit the same web site.

Along with the `401` response, certain other information will be passed back to the client. In particular, it sends a name which is associated with the protected area of the web site. This is called the *realm*, or just the authentication name. The client browser caches the username and password that you supplied, and stores it along with the authentication realm, so that if other resources are requested from the same realm, the same username and password can be returned to authenticate that request without requiring the user to type them in again. This cacheing is usually just for the current browser session, but some browsers allow you to store them permanently, so that you never have to type in your password again.

The authentication name, or realm, will appear in the pop-up box, in order to identify what the username and password are being requested for.

Configuration: Protecting content with basic authentication

There are two configuration steps which you must complete in order to protect a resource using basic authentication. Or three, depending on what you are trying to do.

1. Create a password file
2. Set the configuration to use this password file
3. Optionally, create a group file

Create a password file

In order to determine whether a particular username/password combination is valid, the username and password supplied by the user will need to be compared to some authoritative listing of usernames and password. This is the password file, which you will need to create on the server side, and populate with valid users and their passwords.

Because this file contains sensitive information, it should be stored outside of the document directory. Although, as you will see in a moment, the passwords are encrypted in the file, if a cracker were to gain access to the file, it would be an aid in their attempt to figure out the passwords. And, because people tend to be sloppy with the passwords that they choose, and use the same password for web site authentication as for their bank account, this potentially be a very serious breach of security, even if the content on your web site is not particularly sensitive.

Caution: Encourage your users to use a different password for your web site than for other more essential things. For example, many people tend to use two passwords - one for all of their extremely important things, such as the login to their desktop computer, and for their bank account, and another for less sensitive things, the compromise of which would be less serious.

To create the password file, use the `htpasswd` utility that came with Apache. This will be located in the `bin` directory of wherever you installed Apache. For example, it will probably be located at `/usr/local/apache/bin/htpasswd` if you installed Apache from source.

To create the file, type:

```
htpasswd -c /usr/local/apache/passwd/password username
```

`htpasswd` will ask you for the password, and then ask you to type it again to confirm it:

```
# htpasswd -c /usr/local/apache/passwd/passwords rbowen
New password: mypassword
Re-type new password: mypassword
Adding password for user rbowen
```

Note that in the example shown, a password file is being created containing a user called `rbowen`, and this password file is being placed in the location `/usr/local/apache/passwd/passwords`. You will substitute the location, and the username, which you want to use to start your password file.

If `htpasswd` is not in your path, you will have to type the full path to the file to get it to run. That is, in the example above, you would replace `htpasswd` with `/usr/local/apache/bin/htpasswd`

The `-c` flag is used only when you are creating a new file. After the first time, you will omit the `-c` flag, when you are adding new users to an already-existing password file.

```
htpasswd /usr/local/apache/passwd/passwords sungo
```

The example just shown will add a user named `sungo` to a password file which has already been created earlier. As before, you will be asked for the password at the command line, and then will be asked to confirm the password by typing it again.

Caution: Be very careful when you add new users to an existing password file that you don't use the `-c` flag by mistake. Using the `-c` flag will create a new password file, even if you already have an existing file of that name. That is, it will remove the contents of the file that is there, and replace it with a new file containing only the one username which you were adding.

The password is stored in the password file in encrypted form, so that users on the system will not be able to read the file and immediately determine the passwords of all the users. Nevertheless, you should store the file in as secure a location as possible, with whatever minimum permissions on the file so that the web server itself can read the file. For example, if your server is configured to run as user `nobody` and group `nogroup`, then you should set permissions on the file so that only that user can read the file:

```
chown nobody.nogroup /usr/local/apache/passwd/passwords
chmod 640 /usr/local/apache/passwd/passwords
```

On Windows, a similar precaution should be taken, changing the ownership of the password file to the web server user, so that other users cannot read the file.

Set the configuration to use this password file

Once you have created the password file, you need to tell Apache about it, and tell Apache to use this file in order to require user credentials for admission. This configuration is done with the following directives:

<code>AuthType</code>	Authentication type being used. In this case, it will be set to <code>Basic</code>
<code>AuthName</code>	The authentication realm or name
<code>AuthUserFile</code>	The location of the password file
<code>AuthGroupFile</code>	The location of the group file, if any
<code>Require</code>	The requirement(s) which must be satisfied in order to grant admission

These directives may be placed in a `.htaccess` file in the particular directory being protected, or may go in the main server configuration file, in a `<Directory>` section, or other scope container.

The example shown below defines an authentication realm called `"By Invitation Only"`. The password file located at `/usr/local/apache/passwd/passwords` will be used to verify the user's identity. Only users named `rbowen` or `sungo` will be granted access, and even then only if they provide a password which matches the password stored in the password file.

```
AuthType Basic
AuthName "By Invitation Only"
AuthUserFile /usr/local/apache/passwd/passwords
Require user rbowen sungo
```

The phrase `"By Invitation Only"` will be displayed in the password pop-up box, where the user will have to type their credentials.

You will need to restart your Apache server in order for the new configuration to take effect, if these directives were put in the main server configuration file. Directives placed in `.htaccess` files take effect immediately, since `.htaccess` files are parsed each time files are served.

The next time that you load a file from that directory, you will see the familiar username/password dialog box pop up, requiring that you type the username and password before you are permitted to proceed.

Note that in addition to specifically listing the users to whom you want to grant access, you can specify that any valid user should be let in. This is done with the `valid-user` keyword:

```
Require valid-user
```

Optionally, create a group file

Most of the time, you will want more than one, or two, or even a dozen, people to have access to a resource. You want to be able to define a group of people that have access to that resource, and be able to manage that group of people, adding and removing members, without having to edit the server configuration file, and restart Apache, each time.

This is handled using authentication groups. An authentication group is, as you would expect, a group name associated with a list of members. This list is stored in a group file, which should be stored in the same location as the password file, so that you are able to keep track of these things.

The format of the group file is exceedingly simple. A group name appears first on a line, followed by a colon, and then a list of the members of the group, separated by spaces. For example:

```
authors: rich daniel allan
```

Once this file has been created, you can `Require` that someone be in a particular group in order to get the requested resource. This is done with the `AuthGroupFile` directive, as shown in the following example.

```
AuthType Basic
AuthName "Apache Admin Guide Authors"
AuthUserFile /usr/local/apache/passwd/passwords
AuthGroupFile /usr/local/apache/passwd/groups
Require group authors
```

The authentication process is now one step more involved. When a request is received, and the requested username and password are supplied, the group file is first checked to see if the supplied username is even in the required group. If it is, then the password file will be checked to see if the username is in there, and if the supplied password matches the password stored in that file. If any of these steps fail, access will be forbidden.

Frequently asked questions about basic auth

The following questions tend to get asked very frequently with regard to basic authentication. It should be understood that basic authentication is very basic, and so is limited to the set of features that has been presented above. Most of the more interesting things that people tend to want, need to be implemented using some alternate authentication scheme.

How do I log out?

Since browsers first started implementing basic authentication, website administrators have wanted to know how to let the user log out. Since the browser caches the username and password with the authentication realm, as described earlier in this tutorial, this is not a function of the server configuration, but is a question of getting the browser to forget the credential information, so that the next time the resource is requested, the username and password must be supplied again. There are numerous situations in which this is desirable, such as when using a browser in a public location, and not wishing to leave the browser logged in, so that the next person can get into your bank account.

However, although this is perhaps the most frequently asked question about basic authentication, thus far none of the major browser manufacturers have seen this as being a desirable feature to put into their products.

Consequently, the answer to this question is, you can't. Sorry.

How can I change what the password box looks like?

The dialog that pops up for the user to enter their username and password is ugly. It contains text that you did not indicate that you wanted in there. It looks different in Internet Explorer and Netscape, and contains different text. And it asks for fields that the user

might not understand - for example, Netscape asks the user to type in their ``User ID'', and they might not know what that means. Or, you might want to provide additional explanatory text so that the user has a better idea what is going on.

Unfortunately, these things are features of the browser, and cannot be controlled from the server side. If you want the login to look different, then you will need to implement your own authentication scheme. There is no way to change what this login box looks like if you are using basic authentication.

How to I make it not ask me for my password the next time?

Because most browsers store your password information only for the current browser session, when you close your browser it forgets your username and password. So, when you visit the same web site again, you will need to re-enter your username and password.

There is nothing that can be done about this on the server side.

However, the most recent versions of the major browsers contain the ability to remember your password forever, so that you never have to log in again. While it is debatable whether this is a good idea, since it effectively overrides the entire point of having security in the first place, it is certainly convenient for the user, and simplifies the user experience.

Why does it sometimes ask me for my password twice?

When entering a password-protected web site for the first time, you will occasionally notice that you are asked for your password twice. This may happen immediately after you entered the password the first time, or it may happen when you click on the first link after authenticating the first time.

This happens for a very simple, but nonetheless confusing, reason, again having to do with the way that the browser caches the login information.

Login information is stored on the browser based on the authentication realm, specified by the `AuthName` directive, and by the server name. In this way, the browser can distinguish between the `Private` authentication realm on one site and on another. So, if you go to a site using one name for the server, and internal links on the server refer to that server by a different name, the browser has no way to know that they are in fact the same server.

For example, if you were to visit the URL `http://example.com/private/`, which required authentication, your browser would remember the supplied username and password, associated with the hostname `example.com`. If, by virtue of an internal redirect, or fully-qualified HTML links in pages, you are then sent to the URL `http://www.example.com/private/`, even though this is really exactly the same URL, the browser does not know this for sure, and is forced to request the authentication information again, since `example.com` and `www.example.com` are not exactly the same hostname. Your browser has no particular way to know that these are the same web site.

Security caveat

Basic authentication should not be considered secure for any particularly rigorous definition of secure.

Although the password is stored on the server in encrypted format, it is passed from the client to the server in plain text across the network. Anyone listening with any variety of packet sniffer will be able to read the username and password in the clear as it goes across.

Not only that, but remember that the username and password are passed with every request, not just when the user first types them in. So the packet sniffer need not be listening at a particularly strategic time, but just be listening for long enough to see any request come across the wire.

And, in addition to that, the content itself is also going across the network in the clear, and so if the web site contains sensitive information, the same packet sniffer would have access to that information as it went past, even if the username and password were not used to gain direct access to the web site.

Don't use basic authentication for anything that requires real security. It is a detriment for most users, since very few people will take the trouble, or have the necessary software and/or equipment, to find out passwords. However, if someone had a desire to get in, it would take very little for them to do so.

Digest authentication

Addressing one of the security caveats of basic authentication, digest authentication provides an alternate method for protecting your web content. However, it has a few caveats.

How digest auth works

Digest authentication is implemented by the module `mod_auth_digest`. There is an older module, `mod_digest`, which implemented an older version of the digest authentication specification, but which will probably not work with newer browsers.

Using digest authentication, your password is never sent across the network in the clear, but is always transmitted as an MD5 digest of the user's password. In this way, the password cannot be determined by sniffing network traffic.

The full specification of digest authentication can be seen in the internet standards document RFC 2617, which you can see at <http://www1.ics.uci.edu/pub/ietf/http/rfc2617.txt>. Additional information and resources about MD5 can be found at <http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html>

Configuration: Protecting content with digest authentication

The steps for configuring your server for digest authentication are very similar for those for basic authentication.

1. Create the password file
2. Set the configuration to use this password file
3. Optionally, create a group file

Creating a password file

As with basic authentication, a simple utility is provided to create and maintain the password file which will be used to determine whether a particular user's name and password are valid. This utility is called `htdigest`, and will be located in the `bin` directory of wherever you installed Apache. If you installed Apache from some variety of package manager, `htdigest` is likely to have been placed somewhere in your path.

To create a new digest password file, type:

```
htdigest -c /usr/local/apache/passwd/digest realm username
```

`htdigest` will ask you for the desired password, and then ask you to type it again to confirm it.

Note that the realm for which the authentication will be required is part of the argument list.

Once again, as with basic authentication, you are encouraged to place the generated file somewhere outside of the document directory.

And, as with the `htpasswd` utility, the `-c` flag creates a new file, or, if a file of that name already exists, deletes the contents of that file and generates a new file in its place. Omit the `-c` flag in order to add new user information to an existing password file.

Set the configuration to use this password file

Once you have created a password file, you need to tell Apache about it in order to start using it as a source of authenticated user information. This configuration is done with the following directives:

`AuthType` Authentication type being used. In this case, it will be set to `Digest`

AuthName	The authentication realm or name
AuthDigestFile	The location of the password file
AuthDigestGroupFile	Location of the group file, if any
Require	The requirement(s) which must be satisfied in order to grant admission

These directives may be placed in a `.htaccess` file in the particular directory being protected, or may go in the main server configuration file, in a `<Directory>` section, or another scope container.

The following example defines an authentication realm called "Private". The password file located at `/usr/local/apache/passwd/digest` will be used to verify the user's identity. Only users named `drbacchus` or `dorfl` will be granted access, if they provide a password that matches the password stored in the password file.

```
AuthType Digest
AuthName "Private"
AuthDigestFile /usr/local/apache/passwd/digest
Require user drbacchus dorfl
```

The phrase "Private" will be displayed in the password pop-up box, where the user will have to type their credentials.

Optionally, create a group file

As you have observed, there are not many differences between this configuration process and that required by basic authentication, described in the previous section. This is true also of group functionality. The group file used for digest authentication is exactly the same as that used for basic authentication. That is to say, lines in the group file consist of the name of the group, a colon, and a list of the members of that group. For example:

```
admins: jim roy ed anne
```

Once this file has been created, you can `Require` that someone be in a particular group in order to get the requested resource. This is done with the `AuthDigestGroupFile` directive, as shown in the following example.

```
AuthType Digest
AuthName "Private"
AuthDigestFile /usr/local/apache/passwd/digest
AuthDigestGroupFile /usr/local/apache/passwd/digest.groups
Require group admins
```

The authentication process is the same as that used by basic authentication. It is first verified that the user is in the required group, and, if this is true, then the password is verified.

Caveats

Before you leap into using digest authentication instead of basic authentication, there are a few things that you should know about.

Most importantly, you need to know that, although digest authentication has this great advantage that you don't send your password across the network in the clear, it is not supported by all major browsers in use today, and so you should not use it on a web site on which you cannot control the browsers that people will be using, such as on your intranet site. In particular, Opera 4.0 or later, Microsoft Internet Explorer 5.0 or later, and Amaya support digest authentication, while Netscape, Mozilla, and various other browsers do not.

Next, with regard to security considerations, you should understand two things. Although your password is not passed in the clear, all of your data is, and so this is a rather small measure of security. And, although your password is not really sent at all, but a digest form of it, someone very familiar with the workings of HTTP could use that information - just your digested password - and use that to gain access to the content, since that digested password is really all the information required to access the web site.

The moral of this is that if you have content that really needs to be kept secure, use SSL.

Database authentication modules

Basic authentication and digest authentication both suffer from the same major flaw. They use text files to store the authentication information. The problem with this is that looking something up in a text file is very slow. It's rather like trying to find something in a book that has no index. You have to start at the beginning, and work through it one page at a time until you find what you are looking for. Now imagine that the next time you need to find the same thing, you don't remember where it was before, so you have to start at the beginning again, and work through one page at a time until you find it again. And the next time. And the time after that.

Since HTTP is stateless, authentication has to be verified every time that content is requested. And so every time a document is accessed which is secured with basic or digest authentication, Apache has to open up those text password files and look through them one line at a time, until it finds the user that is trying to log in, and verifies their password. In the worst case, if the username supplied is not in there at all, every line in the file will need to be checked. On average, half of the file will need to be read before the user is found. This is very slow.

While this is not a big problem for small sets of users, when you get into larger numbers of users (where "larger" means a few hundred) this becomes prohibitively slow. In many cases, in fact, valid username/password combinations will get rejected because the authentication module just had to spend so much time looking for the username in the file that Apache will just get tired of waiting and return a failed authentication.

In these cases, you need an alternative, and that alternative is to use some variety of database. Databases are optimized for looking for a particular piece of information in a very large data set. It builds indexes in order to rapidly locate a particular record, and they have query languages for swiftly locating records that match particular criteria.

There are numerous modules available for Apache to authenticate using a variety of different databases. In this section, we'll just look at two modules which ship with Apache.

mod_auth_db and mod_auth_dbm

mod_auth_db and mod_auth_dbm are modules which lets you keep your usernames and passwords in DB or DBM files. There are few practical differences between DB files and DBM files. And, on some operating systems, such as various BSDs, and Linux, they are exactly the same thing. You should pick whichever of the two modules makes the most sense on your particular platform of choice. If you do not have DB support on your platform, you may need to install it. You download an implementation of DB at <http://www.sleepycat.com/>.

Berkeley DB files

DB files, also known as Berkeley database files, are the simplest form of database, and are rather ideally suited for the sort of data that needs to be stored for HTTP authentication. DB files store key/value pairs. That is, the name of a variable, and the value of that variable. While other databases allow the storage of many fields in a given record, a DB file allows only this pairing of key and value.^{21.1} This is ideal for authentication, which requires only the pair of a username and password.

Installing mod_auth_db

```
mod_auth_db!installing
```

For the purposes of this tutorial, we'll talk about installing and configuring mod_auth_db. However, everything that is said here can be directly applied to mod_auth_dbm by simply replacing 'db' with 'dbm' and 'DB' with 'DBM' in the various commands, file names, and directives.

Since mod_auth_db is not compiled in by default, you will need to rebuild Apache in order to get the functionality, unless you built in everything when we started. Note that if you installed Apache with shared object support, you may be able to just build the module and load it in to Apache.

To build Apache from scratch with mod_auth_db built in, use the following `./configure` line in your apache source code directory.

```
./configure --enable-module=auth_db
```

Or, if you had a more complex `configure` command line, you can just add the `-enable-module=auth_db` option to that command line, and you'll get `mod_auth_db` built into your server.

Protecting a directory with `mod_auth_db`

Once you have compiled the `mod_auth_db` module, and loaded it into your web server, you'll find that there's very little difference between using regular authentication and using `mod_auth_db` authentication. The procedure is the same as that we went through with basic and digest authentication:

1. Create the user file.
2. Configure Apache to use that file for authentication.
3. Optionally, create a group file.

Create the user file

The user file for authentication is, this time, not a flat text file, but is a DB file^{21.2}. Fortunately, once again, Apache provides us with a simple utility for the purpose of managing this user file. This time, the utility is called `dbmmanage`, and will be located in the `bin` subdirectory of wherever you installed Apache.

`dbmmanage` is somewhat more complicated to use than `htpasswd` or `htdigest`, but it is still fairly simple. The syntax which you will usually be using is as follows:

```
dbmmanage passwords.db adduser montessor
```

As with `htpasswd`, you will at this point be prompted for a password, and then asked to confirm that password by typing it again. The main difference here is that rather than a text file being created, you are creating a binary file containing the information that you have supplied.

Type `dbmmanage` with no arguments to get the full list of options available with this utility.

Creating your user file with Perl

Note that, if you are so inclined, you can manage your user file with Perl, or any other language which has a DB-file module, for interfacing with this type of database. This covers a number of popular programming languages.

The following Perl code, for example, will add a user 'rbowen', with password 'mypassword', to your password file:

```
use DB_File;
tie %database, 'DB_File', "passwords.dat"
    or die "Can't initialize database: $!\n";

$username = 'rbowen';
$password = 'mypassword';
@chars=(0..9,'a'..'z');
$salt = '', map { $chars[int rand @chars] } (0..1);

$crypt = crypt($password, $salt);
$database{$username} = $crypt;

untie %database;
```

As you can imagine, this makes it very simple to write tools to manage the user and password information stored in these files.

Passwords are stored in Unix `crypt` format, just as they were in the "regular" password files. The 'salt' that is created in the middle there is part of the process, generating a random starting point for that encryption. The technique being used is called a 'tied hash'. The idea is to tie a built-in data structure to the contents of the file, such that when the data structure is changed, the file is

automatically modified at the same time.

Configuration Apache to use this password file

Once you have created the password file, you need to tell Apache about it, and tell Apache to use this file to verify user credentials. This configuration will look almost the same as that for basic authentication. This configuration can go in a `.htaccess` file in the directory to be protected, or can go in the main server configuration, in a `<Directory>` section, or other scope container directive.

The configuration will look something like the following:

```
AuthName "Members Only"
AuthType Basic
AuthDBUserFile /usr/local/apache/passwd/passwords.dat
require user rbowen
```

Now, users accessing the directory will be required to authenticate against the list of valid users who are in `/usr/local/apache/passwd/passwords.dat`.

Optionally, create a group file

As mentioned earlier, DB files store a key/value pair. In the case of group files, the key is the name of the user, and the value is a comma-separated list of the groups to which the user belongs.

While this is the opposite of the way that group files are stored elsewhere, note that we will primarily be looking up records based on the username, so it is more efficient to index the file by username, rather than by the group name.

Groups can be added to your group file using `dbmmanage` and the `add` command:

```
dbmmanage add groupfile rbowen one,two,three
```

In the above example, `groupfile` is the literal name of the group file, `rbowen` is the user being added, and `one`, `two`, and `three` are names of three groups to which this user belongs.

Once you have your groups in the file, you can require a group in the regular way:

```
AuthName "Members Only"
AuthType Basic
AuthDBUserFile /usr/local/apache/passwd/passwords.dat
AuthDBGroupFile /usr/local/apache/passwd/groups.dat
require group three
```

Note that if you want to use the same file for both password and group information, you can do so, but this is a little more complicated to manage, as you have to encrypt the password yourself before you feed it to the `dbmmanage` utility.

Access control

Authentication by username and password is only part of the story. Frequently you want to let people in based on something other than who they are. Something such as where they are coming from. Restricting access based on something other than the identity of the user is generally referred to as *Access Control*.

Allow and Deny

The `Allow` and `Deny` directives let you allow and deny access based on the host name, or host address, of the machine requesting a document. The directive goes hand-in-hand with these is the `Order` directive, which tells Apache in which order to apply the filters.

The usage of these directives is:

```
allow from address
```

where *address* is an IP address (or a partial IP address) or a fully qualified domain name (or a partial domain name).

For example, if you have someone spamming your message board, and you want to keep them out, you could do the following:

```
deny from 205.252.46.165
```

Visitors coming from that address will not be able to see the content behind this directive. If, instead, you have a machine name, rather than an IP address, you can use that.

```
deny from dc.numbersusa.com
```

And, if you'd like to block access from an entire domain, you can specify just part of an address or domain name:

```
deny from 192.101.205
deny from cyberthugs.com
deny from ke
```

Using `Order` will let you be sure that you are actually restricting things to the group that you want to let in, by combining a `deny` and an `allow` directive:

```
Order Deny,Allow
Deny from all
Allow from hostname.example.com
```

Listing just the `allow` directive would not do what you want, because it will let users from that host in, in addition to letting everyone in. What you want is to let in *only* users from that host.

Satisfy

The `Satisfy` directive can be used to specify that several criteria may be considered when trying to decide if a particular user will be granted admission. `Satisfy` can take as an argument one of two options - `all` or `any`. By default, it is assumed that the value is `all`. This means that if several criteria are specified, then all of them must be met in order for someone to get in. However, if set to `any`, then several criteria may be specified, but if the user satisfies any of these, then they will be granted entrance.

A very good example of this is using access control to assure that, although a resource is password protected from outside your network, all hosts inside the network will be given free access to the resource. This would be accomplished by using the `Satisfy` directive, as shown below.

```
<Directory /usr/local/apache/htdocs/sekrit>
  AuthType Basic
  AuthName intranet
  AuthUserFile /www/passwd/users
  AuthGroupFile /www/passwd/groups
  Require group customers
  Allow from internal.com
  Satisfy any
</Directory>
```

In this scenario, users will be let in if they either have a password, or if they are in the internal network.

Summary

The various authentication modules provide a number of ways to restrict access to your host based on the identity of the user. They offer a somewhat standard interface to this functionality, but provide different back-end mechanisms for actually authenticating the user.

And the access control mechanism allows you to restrict access based on criteria unrelated to the identity of the user.

Footnotes

... value [21.1](#)

There are actually a number of implementations that get around this limitation. MLDBM is one of them, for example. However, for the purposes of this discussion, we'll just deal with standard Berkeley DB, which is likely to have shipped with whatever operating system you are already running.

... file [21.2](#)

Or, if you are using `mod_auth_dbm`, a DBM file.



Apache HTTP Server Version 1.3

Dynamic Content with CGI

- [Dynamic Content with CGI](#)
- [Configuring Apache to permit CGI](#)
 - [ScriptAlias](#)
 - [CGI outside of ScriptAlias directories](#)
 - [Explicitly using Options to permit CGI execution](#)
 - [.htaccess files](#)
- [Writing a CGI program](#)
 - [Your first CGI program](#)
- [But it's still not working!](#)
 - [File permissions](#)
 - [Path information](#)
 - [Syntax errors](#)
 - [Error logs](#)
- [What's going on behind the scenes?](#)
 - [Environment variables](#)
 - [STDIN and STDOUT](#)
- [CGI modules/libraries](#)
- [For more information](#)

Dynamic Content with CGI

Related Modules	Related Directives
mod_alias	AddHandler
mod_cgi	Options
	ScriptAlias

The CGI (Common Gateway Interface) defines a way for a web server to interact with external content-generating programs, which are often referred to as CGI programs or CGI scripts. It is the simplest, and most common, way to put dynamic content on your web site. This document will be an introduction to setting up CGI on your Apache web server, and getting started writing CGI programs.

Configuring Apache to permit CGI

In order to get your CGI programs to work properly, you'll need to have Apache configured to permit CGI execution. There are several ways to do this.

ScriptAlias

The `ScriptAlias` directive tells Apache that a particular directory is set aside for CGI programs. Apache will assume that every file in this directory is a CGI program, and will attempt to execute it, when that particular resource is requested by a client.

The `ScriptAlias` directive looks like:

```
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

The example shown is from your default `httpd.conf` configuration file, if you installed Apache in the default location. The `ScriptAlias` directive is much like the `Alias` directive, which defines a URL prefix that is mapped to a particular directory. `Alias` and `ScriptAlias` are usually used for directories that are outside of the `DocumentRoot` directory. The difference between `Alias` and `ScriptAlias` is that `ScriptAlias` has the added meaning that everything under that URL prefix will be considered a CGI program. So, the example above tells Apache that any request for a resource beginning with `/cgi-bin/` should be served from the directory `/usr/local/apache/cgi-bin/`, and should be treated as a CGI program.

For example, if the URL `http://www.example.com/cgi-bin/test.pl` is requested, Apache will attempt to execute the file `/usr/local/apache/cgi-bin/test.pl` and return the output. Of course, the file will have to exist, and be executable, and return output in a particular way, or Apache will return an error message.

CGI outside of ScriptAlias directories

CGI programs are often restricted to `ScriptAlias`'ed directories for security reasons. In this way, administrators can tightly control who is allowed to use CGI programs. However, if the proper security precautions are taken, there is no reason why CGI programs cannot be run from arbitrary directories. For example, you may wish to let users have web content in their home directories with the `UserDir` directive. If they want to have their own CGI programs, but don't have access to the main `cgi-bin` directory, they will need to be able to run CGI programs elsewhere.

Explicitly using Options to permit CGI execution

You could explicitly use the `Options` directive, inside your main server configuration file, to specify that CGI execution was permitted in a particular directory:

```
<Directory /usr/local/apache/htdocs/somedir>  
    Options +ExecCGI  
</Directory>
```

The above directive tells Apache to permit the execution of CGI files. You will also need to tell the server what files are CGI files. The following `AddHandler` directive tells the server to treat all files with the `cgi` or `pl` extension as CGI programs:

```
AddHandler cgi-script cgi pl
```

.htaccess files

A `.htaccess` file is a way to set configuration directives on a per-directory basis. When Apache serves a resource, it looks in the directory from which it is serving a file for a file called `.htaccess`, and, if it finds it, it will apply directives found therein.

`.htaccess` files can be permitted with the `AllowOverride` directive, which specifies what types of directives can appear in these files, or if they are not allowed at all. To permit the directive we will need for this purpose, the following configuration will be needed in your main server configuration:

```
AllowOverride Options
```

In the `.htaccess` file, you'll need the following directive:

```
Options +ExecCGI
```

which tells Apache that execution of CGI programs is permitted in this directory.

Writing a CGI program

There are two main differences between "regular" programming, and CGI programming.

First, all output from your CGI program must be preceded by a MIME-type header. This is HTTP header that tells the client what sort of content it is receiving. Most of the time, this will look like:

```
Content-type: text/html
```

Secondly, your output needs to be in HTML, or some other format that a browser will be able to display. Most of the time, this will be HTML, but occasionally you might write a CGI program that outputs a gif image, or other non-HTML content.

Apart from those two things, writing a CGI program will look a lot like any other program that you might write.

Your first CGI program

The following is an example CGI program that prints one line to your browser. Type in the following, save it to a file called `first.pl`, and put it in your `cgi-bin` directory.

```
#!/usr/bin/perl
print "Content-type: text/html\r\n\r\n";
print "Hello, World.";
```

Even if you are not familiar with Perl, you should be able to see what is happening here. The first line tells Apache (or whatever shell you happen to be running under) that this program can be executed by feeding the file to the interpreter found at the location `/usr/bin/perl`. The second line prints the content-type declaration we talked about, followed by two carriage-return newline pairs. This puts a blank line after the header, to indicate the end of the HTTP headers, and the beginning of the body. The third line prints the string "Hello, World." And that's the end of it.

If you open your favorite browser and tell it to get the address

```
http://www.example.com/cgi-bin/first.pl
```

or wherever you put your file, you will see the one line `Hello, World.` appear in your browser window. It's not very exciting, but once you get that working, you'll have a good chance of getting just about anything working.

But it's still not working!

There are four basic things that you may see in your browser when you try to access your CGI program from the web:

The output of your CGI program

Great! That means everything worked fine.

The source code of your CGI program or a "POST Method Not Allowed" message

That means that you have not properly configured Apache to process your CGI program. Reread the section on [configuring Apache](#) and try to find what you missed.

A message starting with "Forbidden"

That means that there is a permissions problem. Check the [Apache error log](#) and the section below on [file permissions](#).

A message saying "Internal Server Error"

If you check the [Apache error log](#), you will probably find that it says "Premature end of script headers", possibly along with an error message generated by your CGI program. In this case, you will want to check each of the below sections to see what might be preventing your CGI program from emitting the proper HTTP headers.

File permissions

Remember that the server does not run as you. That is, when the server starts up, it is running with the permissions of an unprivileged user - usually ``nobody'', or ``www" - and so it will need extra permissions to execute files that are owned by you. Usually, the way to give a file sufficient permissions to be executed by ``nobody" is to give everyone execute permission on the file:

```
chmod a+x first.pl
```

Also, if your program reads from, or writes to, any other files, those files will need to have the correct permissions to permit this.

The exception to this is when the server is configured to use [suexec](#). This program allows CGI programs to be run under different user permissions, depending on which virtual host or user home directory they are located in. Suexec has very strict permission checking, and any failure in that checking will result in your CGI programs failing with an "Internal Server Error". In this case, you will need to check the suexec log file to see what specific security check is failing.

Path information

When you run a program from your command line, you have certain information that is passed to the shell without you thinking about it. For example, you have a path, which tells the shell where it can look for files that you reference.

When a program runs through the web server as a CGI program, it does not have that path. Any programs that you invoke in your CGI program (like 'sendmail', for example) will need to be specified by a full path, so that the shell can find them when it attempts to execute your CGI program.

A common manifestation of this is the path to the script interpreter (often perl) indicated in the first line of your CGI program, which will look something like:

```
#!/usr/bin/perl
```

Make sure that this is in fact the path to the interpreter.

Syntax errors

Most of the time when a CGI program fails, it's because of a problem with the program itself. This is particularly true once you get the hang of this CGI stuff, and no longer make the above two mistakes. Always attempt to run your program from the command line before you test it via a browser. This will eliminate most of your problems.

Error logs

The error logs are your friend. Anything that goes wrong generates message in the error log. You should always look there first. If the place where you are hosting your web site does not permit you access to the error log, you should probably host your site somewhere else. Learn to read the error logs, and you'll find that almost all of your problems are quickly identified, and quickly solved.

What's going on behind the scenes?

As you become more advanced in CGI programming, it will become useful to understand more about what's happening behind the scenes. Specifically, how the browser and server communicate with one another. Because although it's all very well to write a program that prints ``Hello, World.'', it's not particularly useful.

Environment variables

Environment variables are values that float around you as you use your computer. They are useful things like your path (where the computer searches for the actual file implementing a command when you type it), your username, your terminal type, and so on. For a full list of your normal, every day environment variables, type `env` at a command prompt.

During the CGI transaction, the server and the browser also set environment variables, so that they can communicate with one another. These are things like the browser type (Netscape, IE, Lynx), the server type (Apache, IIS, WebSite), the name of the CGI program that is being run, and so on.

These variables are available to the CGI programmer, and are half of the story of the client-server communication. The complete list of required variables is at <http://hoohoo.ncsa.uiuc.edu/cgi/env.html>

This simple Perl CGI program will display all of the environment variables that are being passed around. Two similar programs are included in the `cgi-bin` directory of the Apache distribution. Note that some variables are required, while others are optional, so you may see some variables listed that were not in the official list. In addition, Apache provides many different ways for you to [add your own environment variables](#) to the basic ones provided by default.

```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
foreach $key (keys %ENV) {
    print "$key --> $ENV{$key}<br>";
}
```

STDIN and STDOUT

Other communication between the server and the client happens over standard input (STDIN) and standard output (STDOUT). In normal everyday context, STDIN means the keyboard, or a file that a program is given to act on, and STDOUT usually means the console or screen.

When you POST a web form to a CGI program, the data in that form is bundled up into a special format and gets delivered to your CGI program over STDIN. The program then can process that data as though it was coming in from the keyboard, or from a file

The ``special format" is very simple. A field name and its value are joined together with an equals (=) sign, and pairs of values are joined together with an ampersand (&). Inconvenient characters like spaces, ampersands, and equals signs, are converted into their hex equivalent so that they don't gum up the works. The whole data string might look something like:

```
name=Rich%20Bowen&city=Lexington&state=KY&sidekick=Squirrel%20Monkey
```

You'll sometimes also see this type of string appended to the a URL. When that is done, the server puts that string into the environment variable called `QUERY_STRING`. That's called a GET request. Your HTML form specifies whether a GET or a POST is used to deliver the data, by setting the `METHOD` attribute in the `FORM` tag.

Your program is then responsible for splitting that string up into useful information. Fortunately, there are libraries and modules available to help you process this data, as well as handle other of the aspects of your CGI program.

CGI modules/libraries

When you write CGI programs, you should consider using a code library, or module, to do most of the grunt work for you. This leads to fewer errors, and faster development.

If you're writing CGI programs in Perl, modules are available on [CPAN](#). The most popular module for this purpose is CGI.pm. You might also consider CGI::Lite, which implements a minimal set of functionality, which is all you need in most programs.

If you're writing CGI programs in C, there are a variety of options. One of these is the CGIC library, from <http://www.boutell.com/cgic/>

For more information

There are a large number of CGI resources on the web. You can discuss CGI problems with other users on the Usenet group comp.infosystems.www.authoring.cgi. And the -servers mailing list from the HTML Writers Guild is a great source of answers to your questions. You can find out more at <http://www.hwg.org/lists/hwg-servers/>

And, of course, you should probably read the CGI specification, which has all the details on the operation of CGI programs. You can find the original version at the [NCSA](#) and there is an updated draft at the [Common Gateway Interface RFC project](#).

When you post a question about a CGI problem that you're having, whether to a mailing list, or to a newsgroup, make sure you provide enough information about what happened, what you expected to happen, and how what actually happened was different, what server you're running, what language your CGI program was in, and, if possible, the offending code. This will make finding your problem much simpler.

Note that questions about CGI problems should **never** be posted to the Apache bug database unless you are sure you have found a problem in the Apache source code.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache Tutorial: Introduction to Server Side Includes

- [Apache Tutorial: Introduction to Server Side Includes](#)
- [What are SSI?](#)
- [Configuring your server to permit SSI](#)
- [Basic SSI directives](#)
 - [Today's date](#)
 - [Modification date of the file](#)
 - [Including the results of a CGI program](#)
- [Additional examples](#)
 - [When was this document modified?](#)
 - [Including a standard footer](#)
 - [What else can I config?](#)
 - [Executing commands](#)
- [Advanced SSI techniques](#)
 - [Setting variables](#)
 - [Conditional expressions](#)
- [Conclusion](#)

Apache Tutorial: Introduction to Server Side Includes

Related Modules	Related Directives
mod_include	Options
mod_cgi	XBitHack
mod_expires	AddType
	AddHandler
	BrowserMatchNoCase

This HOWTO first appeared in Apache Today (<http://www.apachetoday.com/>) as a series of three articles. They appear here by arrangement with ApacheToday and Internet.com.

This article deals with Server Side Includes, usually called simply SSI. In this article, I'll talk about configuring your server to permit SSI, and introduce some basic SSI techniques for adding dynamic content to your existing HTML pages.

In the latter part of the article, we'll talk about some of the somewhat more advanced things that can be done with SSI, such as conditional statements in your SSI directives.

What are SSI?

SSI (Server Side Includes) are directives that are placed in HTML pages, and evaluated on the server while the pages are being served. They let you add dynamically generated content to an existing HTML page, without having to serve the entire page via a CGI program, or other dynamic technology.

The decision of when to use SSI, and when to have your page entirely generated by some program, is usually a matter of how much of the page is static, and how much needs to be recalculated every time the page is served. SSI is a great way to add small pieces of information, such as the current time. But if a majority of your page is being generated at the time that it is served, you need to look for some other solution.

Configuring your server to permit SSI

To permit SSI on your server, you must have the following directive either in your `httpd.conf` file, or in a `.htaccess` file:

```
Options +Includes
```

This tells Apache that you want to permit files to be parsed for SSI directives. Note that most configurations contain multiple [Options](#) directives that can override each other. You will probably need to apply the `Options` to the specific directory where you want SSI enabled in order to assure that it gets evaluated last.

Not just any file is parsed for SSI directives. You have to tell Apache which files should be parsed. There are two ways to do this. You can tell Apache to parse any file with a particular file extension, such as `.shtml`, with the following directives:

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

One disadvantage to this approach is that if you wanted to add SSI directives to an existing page, you would have to change the name of that page, and all links to that page, in order to give it a `.shtml` extension, so that those directives would be executed.

The other method is to use the `XBitHack` directive:

```
XBitHack on
```

`XBitHack` tells Apache to parse files for SSI directives if they have the execute bit set. So, to add SSI directives to an existing page, rather than having to change the file name, you would just need to make the file executable using `chmod`.

```
chmod +x pagename.html
```

A brief comment about what not to do. You'll occasionally see people recommending that you just tell Apache to parse all `.html` files for SSI, so that you don't have to mess with `.shtml` file names. These folks have perhaps not heard about `XBitHack`. The thing to keep in mind is that, by doing this, you're requiring that Apache read through every single file that it sends out to clients, even if they don't contain any SSI directives. This can slow things down quite a bit, and is not a good idea.

Of course, on Windows, there is no such thing as an execute bit to set, so that limits your options a little.

In its default configuration, Apache does not send the last modified date or content length HTTP headers on SSI pages, because these values are difficult to calculate for dynamic content. This can prevent your document from being cached, and result in slower perceived client performance. There are two ways to solve this:

1. Use the `XBitHack Full` configuration. This tells Apache to determine the last modified date by looking only at the date of the originally requested file, ignoring the modification date of any included files.
2. Use the directives provided by [mod_expires](#) to set an explicit expiration time on your files, thereby letting browsers and proxies know that it is acceptable to cache them.

Basic SSI directives

SSI directives have the following syntax:

```
<!--#element attribute=value attribute=value ... -->
```

It is formatted like an HTML comment, so if you don't have SSI correctly enabled, the browser will ignore it, but it will still be visible in the HTML source. If you have SSI correctly configured, the directive will be replaced with its results.

The element can be one of a number of things, and we'll talk some more about most of these in the next installment of this series. For now, here are some examples of what you can do with SSI

Today's date

```
<!--#echo var="DATE_LOCAL" -->
```

The `echo` element just spits out the value of a variable. There are a number of standard variables, which include the whole set of environment variables that are available to CGI programs. Also, you can define your own variables with the `set` element.

If you don't like the format in which the date gets printed, you can use the `config` element, with a `timefmt` attribute, to modify that formatting.

```
<!--#config timefmt="%A %B %d, %Y" -->
Today is <!--#echo var="DATE_LOCAL" -->
```

Modification date of the file

```
This document last modified <!--#flastmod file="index.html" -->
```

This element is also subject to `timefmt` format configurations.

Including the results of a CGI program

This is one of the more common uses of SSI - to output the results of a CGI program, such as everybody's favorite, a ``hit counter."

```
<!--#include virtual="/cgi-bin/counter.pl" -->
```

Additional examples

Following are some specific examples of things you can do in your HTML documents with SSI.

When was this document modified?

Earlier, we mentioned that you could use SSI to inform the user when the document was most recently modified. However, the actual method for doing that was left somewhat in question. The following code, placed in your HTML document, will put such a time stamp on your page. Of course, you will have to have SSI correctly enabled, as discussed above.

```
<!--#config timefmt="%A %B %d, %Y" -->
This file last modified <!--#flastmod file="ssi.shtml" -->
```

Of course, you will need to replace the `ssi.shtml` with the actual name of the file that you're referring to. This can be

inconvenient if you're just looking for a generic piece of code that you can paste into any file, so you probably want to use the `LAST_MODIFIED` variable instead:

```
<!--#config timefmt="%D" -->
This file last modified <!--#echo var="LAST_MODIFIED" -->
```

For more details on the `timefmt` format, go to your favorite search site and look for `ctime`. The syntax is the same.

Including a standard footer

If you are managing any site that is more than a few pages, you may find that making changes to all those pages can be a real pain, particularly if you are trying to maintain some kind of standard look across all those pages.

Using an include file for a header and/or a footer can reduce the burden of these updates. You just have to make one footer file, and then include it into each page with the `include` SSI command. The `include` element can determine what file to include with either the `file` attribute, or the `virtual` attribute. The `file` attribute is a file path, *relative to the current directory*. That means that it cannot be an absolute file path (starting with `/`), nor can it contain `../` as part of that path. The `virtual` attribute is probably more useful, and should specify a URL relative to the document being served. It can start with a `/`, but must be on the same server as the file being served.

```
<!--#include virtual="/footer.html" -->
```

I'll frequently combine the last two things, putting a `LAST_MODIFIED` directive inside a footer file to be included. SSI directives can be contained in the included file, and includes can be nested - that is, the included file can include another file, and so on.

What else can I config?

In addition to being able to `config` the time format, you can also `config` two other things.

Usually, when something goes wrong with your SSI directive, you get the message

```
[an error occurred while processing this directive]
```

If you want to change that message to something else, you can do so with the `errmsg` attribute to the `config` element:

```
<!--#config errmsg="[It appears that you don't know how to use SSI]" -->
```

Hopefully, end users will never see this message, because you will have resolved all the problems with your SSI directives before your site goes live. (Right?)

And you can `config` the format in which file sizes are returned with the `sizefmt` attribute. You can specify `bytes` for a full count in bytes, or `abbrev` for an abbreviated number in Kb or Mb, as appropriate.

Executing commands

I expect that I'll have an article some time in the coming months about using SSI with small CGI programs. For now, here's something else that you can do with the `exec` element. You can actually have SSI execute a command using the shell (`/bin/sh`, to be precise - or the DOS shell, if you're on Win32). The following, for example, will give you a directory listing.

```
<pre>
<!--#exec cmd="ls" -->
</pre>
```

or, on Windows

```
<pre>  
<!--#exec cmd="dir" -->  
</pre>
```

You might notice some strange formatting with this directive on Windows, because the output from `dir` contains the string ``<dir>'' in it, which confuses browsers.

Note that this feature is exceedingly dangerous, as it will execute whatever code happens to be embedded in the `exec` tag. If you have any situation where users can edit content on your web pages, such as with a ``guestbook'', for example, make sure that you have this feature disabled. You can allow SSI, but not the `exec` feature, with the `IncludesNOEXEC` argument to the `Options` directive.

Advanced SSI techniques

In addition to spitting out content, Apache SSI gives you the option of setting variables, and using those variables in comparisons and conditionals.

Caveat

Most of the features discussed in this article are only available to you if you are running Apache 1.2 or later. Of course, if you are not running Apache 1.2 or later, you need to upgrade immediately, if not sooner. Go on. Do it now. We'll wait.

Setting variables

Using the `set` directive, you can set variables for later use. We'll need this later in the discussion, so we'll talk about it here. The syntax of this is as follows:

```
<!--#set var="name" value="Rich" -->
```

In addition to merely setting values literally like that, you can use any other variable, including, for example, environment variables, or some of the variables we discussed in the last article (like `LAST_MODIFIED`, for example) to give values to your variables. You will specify that something is a variable, rather than a literal string, by using the dollar sign (\$) before the name of the variable.

```
<!--#set var="modified" value="$_LAST_MODIFIED" -->
```

To put a literal dollar sign into the value of your variable, you need to escape the dollar sign with a backslash.

```
<!--#set var="cost" value="\$100" -->
```

Finally, if you want to put a variable in the midst of a longer string, and there's a chance that the name of the variable will run up against some other characters, and thus be confused with those characters, you can place the name of the variable in braces, to remove this confusion. (It's hard to come up with a really good example of this, but hopefully you'll get the point.)

```
<!--#set var="date" value="$_{DATE_LOCAL}_$_{DATE_GMT}" -->
```

Conditional expressions

Now that we have variables, and are able to set and compare their values, we can use them to express conditionals. This lets SSI be a tiny programming language of sorts. `mod_include` provides an `if`, `elif`, `else`, `endif` structure for building conditional statements. This allows you to effectively generate multiple logical pages out of one actual page.

The structure of this conditional construct is:

```
<!--#if expr="test_condition" -->
<!--#elif expr="test_condition" -->
<!--#else -->
<!--#endif -->
```

A *test_condition* can be any sort of logical comparison - either comparing values to one another, or testing the "truth" of a particular value. (A given string is true if it is nonempty.) For a full list of the comparison operators available to you, see the `mod_include` documentation. Here are some examples of how one might use this construct.

In your configuration file, you could put the following line:

```
BrowserMatchNoCase macintosh Mac
BrowserMatchNoCase MSIE InternetExplorer
```

This will set environment variables "Mac" and "InternetExplorer" to true, if the client is running Internet Explorer on a Macintosh.

Then, in your SSI-enabled document, you might do the following:

```
<!--#if expr="${Mac} && ${InternetExplorer}" -->
Apologetic text goes here
<!--#else -->
Cool JavaScript code goes here
<!--#endif -->
```

Not that I have anything against IE on Macs - I just struggled for a few hours last week trying to get some JavaScript working on IE on a Mac, when it was working everywhere else. The above was the interim workaround.

Any other variable (either ones that you define, or normal environment variables) can be used in conditional statements. With Apache's ability to set environment variables with the `SetEnvIf` directives, and other related directives, this functionality can let you do some pretty involved dynamic stuff without ever resorting to CGI.

Conclusion

SSI is certainly not a replacement for CGI, or other technologies used for generating dynamic web pages. But it is a great way to add small amounts of dynamic content to pages, without doing a lot of extra work.



Apache HTTP Server Version 1.3

Apache HOWTO documentation

How to:

- [redirect an entire server or directory to a single URL](#)
 - [reset your log files](#)
 - [stop/restrict robots](#)
 - [proxy SSL requests through your non-SSL server](#)
-

How to redirect an entire server or directory to a single URL

There are two chief ways to redirect all requests for an entire server to a single location: one which requires the use of `mod_rewrite`, and another which uses a CGI script.

First: if all you need to do is migrate a server from one name to another, simply use the `Redirect` directive, as supplied by `mod_alias`:

```
Redirect / http://www.apache.org/
```

Since `Redirect` will forward along the complete path, however, it may not be appropriate - for example, when the directory structure has changed after the move, and you simply want to direct people to the home page.

The best option is to use the standard Apache module `mod_rewrite`. If that module is compiled in, the following lines

```
RewriteEngine On
RewriteRule .* http://www.apache.org/ [R]
```

will send an HTTP 302 Redirect back to the client, and no matter what they gave in the original URL, they'll be sent to "http://www.apache.org/".

The second option is to set up a `ScriptAlias` pointing to a **CGI script** which outputs a 301 or 302 status and the location of the other server.

By using a **CGI script** you can intercept various requests and treat them specially, *e.g.*, you might want to intercept **POST** requests, so that the client isn't redirected to a script on the other server which expects POST information (a redirect will lose the POST information.) You might also want to use a CGI script if you don't want to compile `mod_rewrite` into your server.

Here's how to redirect all requests to a script... In the server configuration file,

```
ScriptAlias / /usr/local/httpd/cgi-bin/redirect_script/
```

and here's a simple perl script to redirect requests:

```
#!/usr/local/bin/perl

print "Status: 302 Moved Temporarily\r\n" .
      "Location: http://www.some.where.else.com/\r\n" .
      "\r\n";
```

How to reset your log files

Sooner or later, you'll want to reset your log files (`access_log` and `error_log`) because they are too big, or full of old information you don't need.

`access_log` typically grows by 1Mb for each 10,000 requests.

Most people's first attempt at replacing the logfile is to just move the logfile or remove the logfile. This doesn't work.

Apache will continue writing to the logfile at the same offset as before the logfile moved. This results in a new logfile being created which is just as big as the old one, but it now contains thousands (or millions) of null characters.

The correct procedure is to move the logfile, then signal Apache to tell it to reopen the logfiles.

Apache is signaled using the **SIGHUP** (-1) signal. *e.g.*

```
mv access_log access_log.old
kill -1 `cat httpd.pid`
```

Note: `httpd.pid` is a file containing the **process id** of the Apache httpd daemon, Apache saves this in the same directory as the log files.

Many people use this method to replace (and backup) their logfiles on a nightly or weekly basis.

How to stop or restrict robots

Ever wondered why so many clients are interested in a file called `robots.txt` which you don't have, and never did have?

These clients are called **robots** (also known as crawlers, spiders and other cute names) - special automated clients which wander around the web looking for interesting resources.

Most robots are used to generate some kind of *web index* which is then used by a *search engine* to help locate information.

`robots.txt` provides a means to request that robots limit their activities at the site, or more often than not, to leave the site alone.

When the first robots were developed, they had a bad reputation for sending hundreds/thousands of requests to each site, often resulting in the site being overloaded. Things have improved dramatically since then, thanks to [Guidelines for Robot Writers](#), but even so, some robots may exhibit unfriendly behavior which the webmaster isn't willing to tolerate, and will want to stop.

Another reason some webmasters want to block access to robots, is to stop them indexing dynamic information. Many search engines will use the data collected from your pages for months to come - not much use if you're serving stock quotes, news, weather reports or anything else that will be stale by the time people find it in a search engine.

If you decide to exclude robots completely, or just limit the areas in which they can roam, create a `robots.txt` file; refer to the [robot information pages](#) provided by Martijn Koster for the syntax.

How to proxy SSL requests *through* your non-SSL Apache server (*submitted by David Sedlock*)

SSL uses port 443 for requests for secure pages. If your browser just sits there for a long time when you attempt to access a secure page over your Apache proxy, then the proxy may not be configured to handle SSL. You need to instruct Apache to listen on port 443 in addition to any of the ports on which it is already listening:

```
Listen 80  
Listen 443
```

Then set the security proxy in your browser to 443. That might be it!

If your proxy is sending requests to another proxy, then you may have to set the directive ProxyRemote differently. Here are my settings:

```
ProxyRemote http://nicklas:80/ http://proxy.mayn.franken.de:8080  
ProxyRemote http://nicklas:443/ http://proxy.mayn.franken.de:443
```

Requests on port 80 of my proxy nicklas are forwarded to proxy.mayn.franken.de:8080, while requests on port 443 are forwarded to proxy.mayn.franken.de:443. If the remote proxy is not set up to handle port 443, then the last directive can be left out. SSL requests will only go over the first proxy.

Note that your Apache does NOT have to be set up to serve secure pages with SSL. Proxying SSL is a different thing from using it.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache Tutorials

The following documents give you step-by-step instructions on how to accomplish common tasks with the Apache http server. Many of these documents are located at external sites and are not the work of the Apache Software Foundation. Copyright to documents on external sites is owned by the authors or their assignees. Please consult the [official Apache Server documentation](#) to verify what you read on external sites.

Installation & Getting Started

- [Getting Started with Apache 1.3](#) (ApacheToday)
- [Configuring Your Apache Server Installation](#) (ApacheToday)
- [Getting, Installing, and Running Apache \(on Unix\)](#) (O'Reilly Network Apache DevCenter)
- [Maximum Apache: Getting Started](#) (CNET Builder.com)
- [How to Build the Apache of Your Dreams](#) (Developer Shed)

Basic Configuration

- [An Amble Through Apache Configuration](#) (O'Reilly Network Apache DevCenter)
- [Using .htaccess Files with Apache](#) (ApacheToday)
- [Setting Up Virtual Hosts](#) (ApacheToday)
- [Maximum Apache: Configure Apache](#) (CNET Builder.com)
- [Getting More Out of Apache](#) (Developer Shed)

Security

- [Security and Apache: An Essential Primer](#) (LinuxPlanet)
- [Using User Authentication](#) (Apacheweek)
- [DBM User Authentication](#) (Apacheweek)
- [An Introduction to Securing Apache](#) (Linux.com)
- [Securing Apache - Access Control](#) (Linux.com)
- Apache Authentication [Part 1](#) - [Part 2](#) - [Part 3](#) - [Part 4](#) (ApacheToday)
- [mod_access: Restricting Access by Host](#) (ApacheToday)

Logging

- [Log Rhythms](#) (O'Reilly Network Apache DevCenter)
- [Gathering Visitor Information: Customising Your Logfiles](#) (Apacheweek)
- Apache Guide: Logging [Part 1](#) - [Part 2](#) - [Part 3](#) - [Part 4](#) - [Part 5](#) (ApacheToday)

CGI and SSI

- [Dynamic Content with CGI](#) (ApacheToday)
- [The Idiot's Guide to Solving Perl CGI Problems](#) (CPAN)
- [Executing CGI Scripts as Other Users](#) (LinuxPlanet)
- [CGI Programming FAQ](#) (Web Design Group)
- Introduction to Server Side Includes [Part 1](#) - [Part 2](#) (ApacheToday)
- [Advanced SSI Techniques](#) (ApacheToday)
- [Setting up CGI and SSI with Apache](#) (CNET Builder.com)

Other Features

- [Content Negotiation Explained](#) (Apacheweek)
- [Using Apache Imagemaps](#) (Apacheweek)
- [Keeping Your Images from Adorning Other Sites](#) (ApacheToday)
- [Language Negotiation Notes](#) (Alan J. Flavell)

If you have a pointer to an accurate and well-written tutorial not included here, please let us know by submitting it to the [Apache Bug Database](#).

Apache HTTP Server Version 1.3





Apache HTTP Server

Using Apache With Microsoft Windows

This document explains how to install, configure and run Apache 1.3 under Microsoft Windows. Most of this document assumes that you are installing Windows from a binary distribution. If you want to compile Apache yourself (possibly to help with development, or to track down bugs), see [Compiling Apache for Microsoft Windows](#).

If you find any bugs, please document them on our [bug reporting page](#). Contributions are welcomed, please submit your code or suggestions to the bug report page, or join the new-httpd mailing list.

The bug reporting page and new-httpd mailing list are *not* provided to answer questions about configuration or running Apache. Before you submit a bug report or request, first consult this document, the [Frequently Asked Questions](#) page and the other relevant documentation topics. If you still have a question or problem, post it to the [comp.infosystems.www.servers.ms-windows](#) newsgroup, where many Apache users and several contributions are more than willing to answer new and obscure questions about using Apache on Windows.

[groups.google.com's newsgroup archive](#) offers easy browsing of previous questions. Searching the newsgroup archives, you will usually find your question was already asked and answered by other users!

Warning: Apache on NT has not yet been optimized for performance. Apache still performs best, and is most reliable on Unix platforms. Over time NT performance has improved, and great progress is being made in the upcoming version 2.0 of Apache for the Windows platforms. Folks doing comparative reviews of webserver performance are still asked to compare against Apache on a Unix platform such as Solaris, FreeBSD, or Linux.

-
- [Requirements](#)
 - [Downloading Apache for Windows](#)
 - [Installing Apache for Windows \(binary install\)](#)
 - [Running Apache for Windows](#)
 - [Testing Apache for Windows](#)
 - [Configuring Apache for Windows](#)
 - [Running Apache in a Console Window](#)
 - [Controlling Apache in a Console Window](#)
 - [Running Apache for Windows as a Service](#)
 - [Controlling Apache as a Service](#)
 - [Compiling Apache for Microsoft Windows](#)
-

Requirements

Apache 1.3 is designed to run on Windows NT 4.0 and Windows 2000. The binary installer will only work with the x86 family of processors, such as Intel's. Apache may also run on Windows 95 and 98, but these have not been tested. In all cases TCP/IP networking must be installed.

If running on NT 4.0, installing Service Pack 3 or 6 is recommended, as Service Pack 4 created known issues with TCP/IP and

WinSock integrity that were resolved in Service Pack 5 and later.

Note: "Winsock2" is required for Apache 1.3.7 and later.

If running on Windows 95, the "Winsock2" upgrade must be installed before Apache will run. "Winsock2" for Windows 95 is available [here](#) or via [here](#). Be warned that the Dialup Networking 1.2 (MS DUN) updates include a Winsock2 that is entirely insufficient, and the Winsock2 update must be reinstalled after installing Windows 95 dialup networking. Windows 98, NT (Service Pack 3 or later) and 2000 users need to take no special action, those versions provide Winsock2 as distributed.

Downloading Apache for Windows

Information on the latest version of Apache can be found on the Apache web server at <http://httpd.apache.org/>. This will list the current release, any more recent alpha or beta-test releases, together with details of mirror web and anonymous FTP sites.

You should download the binary build of Apache for Windows named as `apache_1_3_#-win32-src.msi` if you are interested in the source code, or simply `apache_1_3_#-win32-no_src.msi` if you don't plan to do anything with the source code and appreciate a faster download. Each of these files contains the complete Apache runtime. You must have the Microsoft Installer version 1.10 installed on your PC before you can install the Apache runtime distributions. Windows 2000 and Windows ME are both delivered with the Microsoft Installer support, others will need to download it. Instructions on locating the Microsoft Installer, as well as the binary distributions of Apache, are found at <http://httpd.apache.org/dist/httpd/binaries/win32/>

The source code is available in the `-src.msi` distribution, or from the <http://httpd.apache.org/dist/httpd/> distribution directory as a `.zip` file. If you plan on compiling Apache yourself, there is no need to install either `.msi` package. The `.zip` file contains only source code, with MS-DOS line endings (that is `cr/lf` line endings, instead of the single `lf` used for Unix files distributed in `.tar.gz` or `.tar.Z` archives.)

While the source is also available as a `.tar.gz` `.tar.Z` archive, these contain unix `lf` line endings that cause grief for Windows users. To use those archives, you must convert at least the `.mak` and `.dsp` files to have DOS line endings before MSVC can understand them. Please stick with the `.zip` file to spare yourself the headache.

Note: prior to 1.3.17 Apache was distributed as an InstallShield 2.0 `.exe` file. With an increasing number of users unable to run the InstallShield package [on Windows ME or Windows 2000] the binaries were repackaged into the readily available Microsoft Installer `.msi` format.

Installing Apache for Windows

Run the Apache `.msi` file you downloaded above. This will prompt you for:

- whether or not you want to run Apache for all users (installing Apache as a Service), or if you want it installed to run in a console window when you choose the Start Apache shortcut.
- your Server name, Domain name and administrative email account.
- the directory to install Apache into (the default is `C:\Program Files\Apache Group\Apache` although you can change this to any other directory you wish)
- the installation type. The "Complete" option installs everything, including the source code if you downloaded the `-src.msi` package. Choose the "Custom" install if you choose not to install the documentation, or the source code from that package.

During the installation, Apache will configure the files in the `conf` directory for your chosen installation directory. However if any of the files in this directory already exist they will **not** be overwritten. Instead the new copy of the corresponding file will be left with the extension `.default.conf`. So, for example, if `conf\httpd.conf` already exists it will not be altered, but the version which would have been installed will be left in `conf\httpd.default.conf`. After the installation has finished you should manually check to see what is new in the `.default.conf` file, and if necessary update your existing configuration files.

Also, if you already have a file called `htdocs\index.html` then it will not be overwritten (no `index.html.default` file will be installed either). This should mean it is safe to install Apache over an existing installation (but you will have to stop the existing server running before doing the installation, then start the new one after the installation is finished).

After installing Apache, you should edit the configuration files in the `conf` directory as required. These files will be configured during the install ready for Apache to be run from the directory where it was installed, with the documents served from the subdirectory `htdocs`. There are lots of other options which should be set before you start really using Apache. However to get started

quickly the files should work as installed.

If you eventually uninstall Apache, your configuration and log files will not be removed. You will need to delete the installation directory tree ("C:\Program Files\Apache Group" by default) yourself if you do not care to keep your configuration and other web files. Since the httpd.conf file is your accumulated effort in using Apache, you need to take the effort to remove it. The same happens for all other files you may have created, as well as any log files Apache created.

Running Apache for Windows

There are two ways you can run Apache:

- As a ["service"](#). This is the best option if you want Apache to automatically start when your machine boots, and to keep Apache running when you log-off.
- From a [console window](#). Closing this console window will terminate the Apache server.

Complete the steps below before you attempt to start Apache as a Windows ["service"](#)!

To run Apache from a console window, select the "Start Apache as console app" option from the Start menu (in Apache 1.3.4 and earlier, this option was called "Apache Server"). This will open a console window and start Apache running inside it. The window will remain active until you stop Apache. To stop Apache running, either press select the "Shutdown Apache console app" icon option from the Start menu (this is not available in Apache 1.3.4 or earlier), or see [Controlling Apache in a Console Window](#) for commands to control Apache in a console window.

In Apache 1.3.13 and above it is now quite safe to press Ctrl+C or Ctrl+Break to stop the Apache in the console window. And on Windows NT/2000 with version 1.3.13, Apache will stop if you select 'Close' from the system menu (clicking the icon on the top-left corner of the console window) or click the close (X) button on the top-right corner. The Close menu item and close (X) button also work on Windows 95/98 as of Apache version 1.3.15. But do *not* try any of these approaches on earlier versions of the Apache server, since Apache would not clean up.

Testing Apache for Windows

If you have trouble starting Apache please use the following steps to isolate the problem. This applies if you started Apache using the "Start Apache as a console app" shortcut from the Start menu and the Apache console window closes immediately (or unexpectedly) or if you have trouble starting Apache as a service.

Run the "Command Prompt" from the Start Menu - Programs list. Change to the folder to which you installed Apache, type the command `apache`, and read the error message. Then review the `error.log` file for configuration mistakes. If you accepted the defaults when you installed Apache, the commands would be:

```
c:
cd "\program files\apache group\apache"
apache
Wait for Apache to exit, or press Ctrl+C
more <logs\error.log
```

After looking at the `error.log` you will probably have a good chance of working out what went wrong and be able to fix the problem and try again. If you are unable to work it out then please follow the [guidelines for assistance](#) at the top of this document or in the [FAQ](#). Many users discover that the nature of the `httpd.conf` file is easier to manage and audit than page after page of configuration dialog boxes.

After starting Apache running (either in a console window or as a service) it will be listening to port 80 (unless you changed the `Port`, `Listen` or `BindAddress` directives in the configuration files). To connect to the server and access the default page, launch a browser and enter this URL:

```
http://localhost/
```

This should respond with a welcome page, and a link to the Apache manual. If nothing happens or you get an error, look in the `error.log` file in the `logs` directory. If your host isn't connected to the net, you may have to use this URL:

`http://127.0.0.1/`

Once your basic installation is working, you should configure it properly by editing the files in the conf directory.

Because Apache *CANNOT* share the same port with another TCP/IP application, you may need to stop or uninstall certain services first. These include (but are not limited to) other web servers, and firewall products such as BlackIce. If you can only start Apache with these services disabled, reconfigure either Apache or the other product so that they do not listen on the same TCP/IP ports. You may find the Windows "netstat -an" command useful in finding out what ports are in use.

Configuring Apache for Windows

Apache is configured by files in the conf directory. These are the same as files used to configure the Unix version, but there are a few different directives for Apache on Windows. See the [Apache documentation](#) for all the available directives.

Begin configuring the Apache server by reviewing `httpd.conf` and its directives. Although the files `access.conf` and `srml.conf` both exist, these are old files which are no longer used by most administrators, and you will find no directives there.

`httpd.conf` contains a great deal of documentation itself, followed by the default configuration directives recommended when starting with the Apache server. Begin by reading these comments to understand the configuration file, and make small changes, starting Apache in a console window with each change. If you make a mistake, it will be easier to back up to configuration that last worked. You will have a better idea of which change caused the server to fail.

The main differences in Apache for Windows are:

- Because Apache for Windows is multithreaded, it does not use a separate process for each request, as Apache does with Unix. Instead there are usually only two Apache processes running: a parent process, and a child which handles the requests. Within the child each request is handled by a separate thread. So, "process"-management directives are different:
 - [MaxRequestsPerChild](#) - Like the Unix directive, this controls how many requests a process will serve before exiting. However, unlike Unix, a process serves all the requests at once, not just one, so if this is set, it is recommended that a very high number is used. The recommended default, `MaxRequestsPerChild 0`, does not cause the process to ever exit.
 - [ThreadsPerChild](#) - This directive is new, and tells the server how many threads it should use. This is the maximum number of connections the server can handle at once; be sure and set this number high enough for your site if you get a lot of hits. The recommended default is `ThreadsPerChild 50`.
- The directives that accept filenames as arguments now must use Windows filenames instead of Unix ones. However, because Apache uses Unix-style names internally, you must use forward slashes, not backslashes. Drive letters can be used; if omitted, the drive with the Apache executable will be assumed.
- Apache for Windows has the ability to load modules at runtime, without recompiling the server. If Apache is compiled normally, it will install a number of optional modules in the `\modules` directory. To activate these, or other modules, the new [LoadModule](#) directive must be used. For example, to active the status module, use the following (in addition to the status-activating directives in `access.conf`):

```
LoadModule status_module modules/mod_status.so
```

Information on [creating loadable modules](#) is also available. Note that some 3rd party modules may be distributed in the old style names, `ApacheModuleFoo.dll`. Always set the `LoadModule` command as directed as documented by the 3rd party module's own documentation.

- Apache for Windows version 1.3 series is implemented in synchronous calls. This poses an enormous problem for CGI authors, who won't see unbuffered results sent immediately to the browser. This is not the behavior described for CGI in Apache, but it is a side-effect of the Windows port. Apache 2.0 is making progress to implement the expected asynchronous behavior, and we hope to discover that the NT/2000 implementation allows CGI's to behave as documented.
- Apache can also load ISAPI Extensions (*i.e.*, Internet Server Applications), such as those used by Microsoft's IIS, and other Windows servers. [More information is available](#). Note that Apache *CANNOT* load ISAPI Filters.
- When running CGI scripts, the method Apache uses to find the interpreter for the script is configurable using the [ScriptInterpreterSource](#) directive.

- Since it is often difficult to manage files with names like `.htaccess` under windows, you may find it useful to change the name of this configuration file using the [AccessFilename](#) directive.

Running Apache in a Console Window

The Start menu icons and the NT Service manager can provide a simple interface for administering Apache. But in some cases it is easier to work from the command line.

When working with Apache it is important to know how it will find the configuration files. You can specify a configuration file on the command line in two ways:

- `-f` specifies a path to a particular configuration file:

```
apache -f "c:\my server\conf\my.conf"
```

```
apache -f test\test.conf
```

- `-n` specifies the configuration file of an installed Apache service (Apache 1.3.7 and later):

```
apache -n "service name"
```

In these cases, the proper `ServerRoot` should be set in the configuration file.

If you don't specify a configuration file name with `-f` or `-n`, Apache will use the file name compiled into the server, usually `conf/httpd.conf`. Invoking Apache with the `-V` switch will display this value labeled as `SERVER_CONFIG_FILE`. Apache will then determine its `ServerRoot` by trying the following, in this order:

- A `ServerRoot` directive via a `-C` switch.
- The `-d` switch on the command line.
- The current working directory
- A registry entry, created if you did a binary install.
- The server root compiled into the server.

The server root compiled into the server is usually `/apache`. Invoking Apache with the `-V` switch will display this value labeled as `HTTPD_ROOT`.

When invoked from the start menu, Apache is usually passed no arguments, so using the registry entry is the preferred technique for console Apache.

During a binary installation, a registry key will have been installed, for example:

```
HKEY_LOCAL_MACHINE\Software\Apache Group\Apache\1.3.13\ServerRoot
```

This key is compiled into the server and can enable you to test new versions without affecting the current version. Of course you must take care not to install the new version on top of the old version in the file system.

If you did not do a binary install then Apache will in some scenarios complain about the missing registry key. This warning can be ignored if it otherwise was able to find its configuration files.

The value of this key is the `"ServerRoot"` directory, containing the `conf` directory. When Apache starts it will read the `httpd.conf` file from this directory. If this file contains a `ServerRoot` directive which is different from the directory obtained from the registry key above, Apache will forget the registry key and use the directory from the configuration file. If you copy the Apache directory or configuration files to a new location it is vital that you update the `ServerRoot` directory in the `httpd.conf` file to the new location.

To run Apache from the command line as a console application, use the following command:

```
apache
```

Apache will execute, and will remain running until it is stopped by pressing control-C.

Controlling Apache in a Console Window

You can tell a running Apache to stop by opening another console window and running:

```
apache -k shutdown
```

Note: This option is only available with Apache 1.3.3 and later.

For earlier versions, you must use Control-C in the Apache console window to shut down the server.

From version 1.3.3 through 1.3.12, this should be used instead of pressing Control-C in a running Apache console window, because it allowed Apache to end any current transactions and cleanup gracefully.

As of version 1.3.13 pressing Control-C in the running window will cleanup Apache quite gracefully, and you may use -k stop as an alias for -k shutdown. Earlier versions do not understand -k stop.

You can also tell Apache to restart. This makes it re-read the configuration files. Any transactions in progress are allowed to complete without interruption. To restart Apache, run:

```
apache -k restart
```

Note: This option is only available with Apache 1.3.3 and later. For earlier versions, you need to use Control-C in the Apache console window to shut down the server, and then restart the server with the Apache command.

Another *very useful* feature is the configuration files test option. To test the Apache configuration files, run:

```
apache -t
```

This is especially useful following alterations to the configuration files while Apache is still running. You can make the changes, confirm that the syntax is good by issuing the "apache -t" command, then restart Apache with "apache -k restart". Apache will re-read the configuration files, allowing any transactions in progress to complete without interruption. Any new request will then be served using the new configuration.

Note: for people familiar with the Unix version of Apache, these commands provide a Windows equivalent to `kill -TERM pid` and `kill -USR1 pid`. The command line option used, -k, was chosen as a reminder of the "kill" command used on Unix.

Apache HTTP Server





Apache HTTP Server

Compiling Apache for Microsoft Windows

There are many important points before you begin compiling Apache. See [Using Apache with Microsoft Windows](#) before you begin.

Compiling Apache requires Microsoft Visual C++ 5.0 or 6.0 to be properly installed. It can be built with command-line tools, or within the Visual Studio environment. Consult the VC++ manual to determine how to install them. Be especially aware that the `vcvars32.bat` file from the Program Files/DevStudio/VC/bin folder, and the `setenv.bat` file from the Platform SDK, may be required to prepare the command-line tools for command-line builds (e.g. using `nmake`). To install apache with the `Makefile.win` or the `InstallBin` project in the Visual Studio IDE, the `awk` utility is also required.

First, you should install `awk.exe` where it can be found in the path and the DevStudio environment, if you plan to use the IDE. There are many versions of `awk` available for Windows; the easiest to install is available from Brian Kernighan's <http://cm.bell-labs.com/cm/cs/who/bwk/> site. When downloading <http://cm.bell-labs.com/cm/cs/who/bwk/awk95.exe> from this site, you must save it with the name `awk.exe` rather than `awk95.exe`.

Note that DevStudio will only find `awk.exe` if its location is listed under the Tools menu Options... Directories settings for the Executable files. Add the path where `awk.exe` resides to this list, as needed.

Then unpack the Apache distribution into an appropriate directory. Open a command-line prompt, and change to the `src` subdirectory of the Apache distribution.

The master Apache makefile instructions are contained in the `Makefile.win` file. To compile Apache on Windows NT, simply use one of the following commands:

- `nmake /f Makefile.win _apacher` (release build)
- `nmake /f Makefile.win _apached` (debug build)

These will both compile Apache. The latter will include debugging information in the resulting files, making it easier to find bugs and track down problems.

If you get an error such as "the name specified is not recognized..." then you need to run `vcvars32.bat` first. Enter the following command;

```
"c:\Program Files\DevStudio\VC\Bin\VCVARS32.BAT"
```

(you will need to adjust this command so it matches the directory where your VC was installed.)

If you are a Visual C++ 5.0 user, and have installed a recent Platform SDK, you may also need to enter the following command (adjusted for the install directory of the Platform SDK update);

```
"c:\Program Files\Platform SDK\SETENV.BAT"
```

Then try the `nmake` command again.

Note that the Windows Platform SDK update is required to enable all supported `mod_isapi` features. The SDK files distributed with Microsoft Visual C++ 5.0 are out of date. Without a recent update, Apache will issue warnings under MSVC++ 5.0 that some `mod_isapi` features will be disabled. Look for the update at <http://msdn.microsoft.com/platformsdk/>.

Apache can also be compiled using VC++'s Visual Studio development environment. To simplify this process, a Visual Studio workspace, `Apache.dsw`, is provided in the `src` folder. This workspace exposes the entire list of working `.dsp` projects that are

required for the complete Apache binary release. It includes dependencies between the projects to assure that they are built in the appropriate order. InstallBin is the top-level project that will build all other projects, and install the compiled files into their proper locations.

These .dsp project files are distributed in Visual C++ 6.0 format. Visual C++ 5.0 (97) will recognize them with the single exception of the /ZI flag, which corresponds to the VC 5.0 /Zi flag for debugging symbols. To quickly prepare the .dsp files for the Visual Studio 5.0 (97), you can use the perl scripts distributed in the src\helpers folder:

```
cd src\helpers
cvstodsp5.pl
```

This command assumes you have a Perl interpreter installed and registered for files of type .pl. The list of converted .dsp project files will be displayed as they are converted. If you contribute back a patch that offers revised project files, please convert them back with the script dsp5tocvs.pl, which puts the projects back to Visual Studio 6.0 format.

The core .dsp projects built by Apache.dsw and makefile.win are:

- os\win32\ApacheOS.dsp
- os\win32\Win9xConHook.dsp
- regex\regex.dsp
- ap\ap.dsp
- lib\expat-lite\xmltok.dsp
- lib\expat-lite\xmlparse.dsp *requires xmltok*
- lib\sdbm.dsp
- main\gen_uri_delims.dsp
- main\gen_test_char.dsp
- ApacheCore.dsp *requires all of the above*
- Apache.dsp *requires ApacheCore*

In addition, the os\win32 subdirectory contains project files for the optional modules, all of which require ApacheCore.

- os\win32\mod_auth_anon.dsp
- os\win32\mod_auth_dbm.dsp *also requires sdbm*
- os\win32\mod_auth_digest.dsp
- os\win32\mod_cern_meta.dsp
- os\win32\mod_digest.dsp
- os\win32\mod_expires.dsp
- os\win32\mod_headers.dsp
- os\win32\mod_info.dsp
- os\win32\mod_mime_magic.dsp
- os\win32\mod_proxy.dsp
- os\win32\mod_rewrite.dsp
- os\win32\mod_speling.dsp
- os\win32\mod_status.dsp
- os\win32\mod_unique_id.dsp
- os\win32\mod_usertrack.dsp
- os\win32\mod_vhost_alias.dsp

The support\ folder contains project files for additional programs that are not part of the Apache runtime, but are used by the administrator to maintain password and log files.

- support\htdigest.dsp
- support\htpasswd.dsp

- support\logresolve.dsp
- support\rotatelogs.dsp

Once Apache has been compiled, it needs to be installed in its server root directory. The default is the \Apache directory, on the current hard drive.

To install the files into the c:\ServerRoot directory automatically, use one the following nmake commands (see above):

- nmake /f Makefile.win installr INSTDIR=c:\ServerRoot (for release build)
- nmake /f Makefile.win installd INSTDIR=c:\ServerRoot (for debug build)

The c:\ServerRoot argument to INSTDIR gives the installation directory (it can be omitted if Apache is to be installed into \Apache).

This will install the following:

- c:\ServerRoot\Apache.exe - Apache program
- c:\ServerRoot\ApacheCore.dll - Apache runtime [shared library]
- c:\ServerRoot\Win9xConHook.dll - Win9x console fixups [shared library]
- c:\ServerRoot\xmlparse.dll - XML parser [shared library]
- c:\ServerRoot\xmltok.dll - XML token engine [shared library]
- c:\ServerRoot\bin*.exe - Administration programs
- c:\ServerRoot\cgi-bin - Example CGI scripts
- c:\ServerRoot\conf - Configuration files directory
- c:\ServerRoot\icons - Icons for FancyIndexing
- c:\ServerRoot\include*.h - Apache header files
- c:\ServerRoot\htdocs - Welcome index.html pages
- c:\ServerRoot\htdocs\manual - Apache documentation
- c:\ServerRoot\lib - Static library files
- c:\ServerRoot\libexec - Dynamic link libraries
- c:\ServerRoot\logs - Empty logging directory
- c:\ServerRoot\modules\mod_*.dll - Loadable Apache modules

If you do not have nmake, or wish to install in a different directory, be sure to use a similar naming scheme.

To simplify the process, dependencies between all projects are defined in the Microsoft Visual Studio workspace file:

```
src/Apache.dsw
```

This assures that lower-level sources are rebuilt from within Visual Studio. The top level project is InstallBin, which invokes Makefile.win to move the compiled executables and dlls. You may personalize the INSTDIR= setting by changing the Settings for InstallBin, Build command line entry under the General tab. The default from within the InstallBin.dsp project is one level up (..) from the src tree. Modify the InstallBin settings and edit the INSTDIR=.. entry to the desired target directory.

Apache HTTP Server





Apache HTTP Server

Running Apache for Windows as a Service

Apache can be run as a service on Windows NT/2000. (There is also some experimental support for similar behavior on [Windows 95/98](#), introduced with Apache 1.3.13).

Installing Apache as a service should only be done once you can successfully run it in a console window. See [Using Apache with Microsoft Windows](#) before you attempt to install or run Apache as a service. Changes to the httpd.conf file should always be followed by starting Apache as a console window. If this succeeds, the service should succeed.

NOTE: Prior to version 1.3.13, the configuration was *not tested* prior to performing the installation, and a lack of service dependencies often caused the console window to succeed, but the service would still fail. See [below](#) if you are having problems running a version of Apache prior to 1.3.13 to resolve the issue. If you have this problem with version 1.3.13 or greater, first try uninstalling (-u) and re-installing (-i) the Apache service. Better yet, upgrade to the most recent version.

To start Apache as a service, you first need to install it as a service. Multiple Apache services can be installed, each with a different name and configuration. To install the default Apache service named "Apache", choose the "Install as Service for All Users" option when launching the Apache installation package. Once this is done you can start the "Apache" service by opening the Services window (in the Control Panel, hidden in "Administrative Tools" on Windows 2000), selecting Apache, then clicking on Start. Apache will now be running, hidden in the background. You can later stop Apache by clicking on Stop. As an alternative to using the Services window, you can start and stop the "Apache" service from the command line with

```
NET START APACHE
NET STOP APACHE
```

See [Controlling Apache as a Service](#) for more information on installing and controlling Apache services.

Apache, unlike many other Windows NT/2000 services, logs most errors to its own error.log file, in the logs folder within the Apache server root folder. You will find few Apache error details in the Windows NT Event Log. Only errors as Apache attempts to start are captured in the Application Event Log.

After starting Apache as a service (or if you have trouble starting it) you can test it using the same [procedure](#) as for running in a console window. Remember to use the command:

```
apache -n "service name"
```

to assure you are using the service's default configuration.

Running Apache for Windows as a Service

Note: The -n option to specify a service name is only available with Apache 1.3.7 and later. Earlier versions of Apache only support the default service name 'Apache'. Only version 1.3.21 supports Windows 2000 ability to 'rename' the display name of a service.

You can install Apache as a Windows NT service as follows:

```
apache -i -n "service name"
```

To install a service to use a particular configuration, specify the configuration file when the service is installed:

```
apache -i -n "service name" -f "\my server\conf\my.conf"
```

You can pass any other arguments, such as `-d` to change the default server root directory, `-D`, `-C` or `-c` to change config file processing, etc. Since these are stored in the registry and are difficult to modify, use this command to clear the options and replace them with a new list of options;

```
apache -k config -n "service name" -f "\my server\conf\my.conf"
```

To remove an Apache service, use:

```
apache -u -n "service name"
```

The default "service name", if one is not specified, is "Apache".

Once a service is installed, you can use the `-n` option, in conjunction with other options, to refer to a service's configuration file. For example:

To test a service's configuration file and report the default options for the service (configured with `-i`, `-k install` or `-k config`) use:

```
apache -n "service name" -t
```

To start a console Apache using a service's configuration file and its default options, use:

```
apache -n "service name"
```

Effective with Apache release 1.3.15, the `-k install` option was added as an alias to `-i`, and the `-k uninstall` option was added as an alias to `-u`. The original `-i` and `-u` options are deprecated in Apache 2.0. These aliases were added to ease the transition for administrators running both versions.

Important Note on service dependencies:

Prior to Apache release 1.3.13, the dependencies required to successfully start an installed service were not configured. After installing a service using earlier versions of Apache, you must follow these steps:

```
Run regedt32
Select Window - "HKEY_LOCAL_MACHINE on Local Machine" from the menu
Double-click to open the SYSTEM, then the CurrentControlSet keys
Scroll down and click on the Apache servicename
Select Edit - Add Value... from the menu
Fill in the Add Value dialog with
    Value Name: DependOnGroup
    Data Type: REG_MULTI_SZ
    and click OK
Leave the Multi-String Editor dialog empty and click OK
Select Edit - Add Value... from the menu
Fill in the Add Value dialog with
    Value Name: DependOnService
    Data Type: REG_MULTI_SZ
    and click OK
Type the following list (one per line) in the Multi-String Editor dialog
    Tcpip
    Afd
    and click OK
```

If you are using COM or DCOM components from a third party module, ISAPI, or other add-in scripting technologies such as ActiveState Perl, you may also need to add the entry `Rpcss` to the `DependOnService` list. To avoid exposing the TCP port 135 when it is unnecessary, Apache does not create that entry upon installation. Follow the directions above to find or create the

DependOnService value, double click that value if it already exists, and add the Rpscs entry to the list.

Other installations may require additional dependencies. If any files required at startup reside on a network drive, you may need to first configure the service to depend on the network redirector (usually lanmanworkstation) and follow the directions below under the "User Account for Apache Service to Run As" section.

Attempting to use both IIS and Apache, on the same port but on two different IP addresses may require the W3SVC (IIS) to start prior to Apache, because IIS tries to glom onto all IP addresses (0.0.0.0) as it starts, and will fail if Apache has already started on a specific IP address.

Apache 1.3.21 introduces a MUCH simpler way to add a dependency. The -W "servicename" argument modifies the -k install or -k config commands to configure a dependency for Apache. Multiple -W arguments may be given, but they should all occur after the -k option. For example, to add the LanmanWorkstation dependency to an installed "Apache" service, use this command:

```
apache -k config -n Apache -W LanmanWorkstation
```

User Account for Apache Service to Run As (NT/2000)

When Apache is first installed as a service (e.g. with the -i option) it will run as user "System" (the LocalSystem account). There should be few issues if all resources for the web server reside on the local system, but it has broad security privileges to affect the local machine!

LocalSystem is a very privileged account locally, so you shouldn't run any shareware applications there. However, it has no network privileges and cannot leave the machine via any NT-secured mechanism, including file system, named pipes, DCOM, or secure RPC.

NEVER grant network privileges to the SYSTEM account! Create a new user account instead, grant the appropriate privileges to that user, and use the 'Log On As:' option. Select the Start Menu -> Settings -> Control Panel -> Services -> apache service ... and click the "Startup" button to access this setting.

A service that runs in the context of the LocalSystem account inherits the security context of the SCM. It is not associated with any logged-on user account and does not have credentials (domain name, user name, and password) to be used for verification.

The SYSTEM account has no privileges to the network, so shared pages or a shared installation of Apache is invisible to the service. If you intend to use *any* network resources, the following steps should help:

- Select Apache from the Control Panel's Service dialog and click Startup.
- Verify that the service account is correct. You may wish to create an account for your Apache services.
- Retype the password and password confirmation.
- Go to User Manager for Domains.
- Click on Policies from the title bar menu, and select User Rights.
- Select the option for Advanced User Rights.
- In the drop-down list, verify that the following rights have been granted to the selected account:
 - Act as part of the operating system
 - Back up files and directories
 - Log on as a service
 - Restore files and directories
- Confirm that the selected account is a member of the Users group.
- Confirm the selected account has access to all document and script directories (minimally read and browse access).
- Confirm the selected account has read/write/delete access to the Apache logs directory!

If you allow the account to log in as a user, then you can log in yourself and test that the account has the privileges to execute the scripts, read the web pages, and that you can start Apache in a console window. If this works, and you have followed the steps above, Apache should execute as a service with no problems.

Note: error code 2186 is a good indication that you need to review the 'Log On As' configuration, since the server can't access a

required network resource.

Troubleshooting Apache for Windows as a Service

When starting Apache as a service you may encounter an error message from Windows service manager. For example if you try to start Apache using the Services applet in Windows Control Panel you may get the following message;

```
Could not start the apache service on \\COMPUTER
Error 1067; The process terminated unexpectedly.
```

You will get this error if there is any problem starting Apache. In order to see what is causing the problem you should follow the instructions for [Testing Apache at the Command Prompt](#).

Also, Apache 1.3.13 now records startup errors in the Application Event Log under Windows NT/2000, if Apache is run as a service. Run the Event Viewer and select Log ... Application to see these events.

Check the Application Event Log with the Event Viewer in case of any problems, even if no error message pops up to warn you that an error occurred.

Running Apache for Windows from the Command Line

For details on controlling Apache service from the command line, please refer to [console command line](#) section.

Controlling Apache as a Service

Multiple instances of Apache can be installed and run as services. Signal an installed Apache service to start, restart, or shutdown/stop as follows:

```
apache -n "service name" -k start
apache -n "service name" -k restart
apache -n "service name" -k shutdown
apache -n "service name" -k stop
```

For the default "Apache" service, the -n Apache option is still required, since the -k commands without the -n option are directed at Apache running in a console window. The quotes are only required if the service name contains spaces.

Note: the -k stop alias for the -k shutdown command was introduced in Apache version 1.3.13. Earlier versions of Apache will only recognize the -k shutdown option. Prior to 1.3.3, Apache did not recognize *any* -k options at all!

Note that you may specify startup options on the apache -k start command line, including the -D, -C and -c options. These affect the processing of the service configuration, and may be used with an <IfDefine> block to conditionally process directives. You may also override the server root path or configuration file with the -d or -f options. The options should also be passed to the -k restart command, but they are ignored if the service is running, and only processed if the service is started.

The service also appears in the Service Control applet on Windows NT/2000. For NT, this is found in the Settings -> Control Panel -> Services entry, and on 2000 it is found in the Settings -> Control Panel -> Administrative Tools -> Services entry. Here you can select the desired Apache service to start or stop it. Pass additional options such as -D, -C and -c, or override the default -d or -f options in the Start Parameters box before clicking the Start button. These options behave identically to the apache -k start command.

In addition, you can use the native Windows NT/2000 command NET to start and stop Apache services:

```
NET START "service name"
NET STOP "service name"
```

Again, quotes are only required if the service name contains spaces. There is no way using the NET START command to pass additional options such as -D, -c or -C using the NET START command. If options are required, use one of the other two methods

instead.

Experimental Windows 95/98 Service

Note: The service options for Windows 95 and 98 are only available with Apache 1.3.13 and later. Earlier versions of Apache only supported Apache in a console window for Windows 95/98.

There is some support for Apache on Windows 95/98 to behave in a similar manner as a service on Windows NT/2000. It is *experimental*, if it works (at all) the Apache Software Foundation will not attest to its reliability or future support. Proceed at your own risk!

Once you have confirmed that Apache runs correctly at the [Command Prompt](#) you can install, control and uninstall it with the same commands as the Windows NT/2000 version.

There are, however, significant differences that you should note:

Apache will attempt to start and if successful it will run in the background. If you run the command

```
Apache -n "service name" -k start
```

via a shortcut on your desktop, for example, then if the service starts successfully a console window will flash up but immediately disappears. If Apache detects any errors on startup such as a incorrect entries in the httpd.conf file, then the console window will remain visible. This may display an error message which will be useful in tracking down the cause of the problem, and you should also review the error.log file in the Apache logs directory.

Windows 95/98 does not support NET START or NET STOP commands so you must use Apache's Service Control options at a command prompt. You may wish to set up a shortcut for each of these commands so that you can just choose it from the start menu or desktop to perform the required action.

Apache and Windows 95/98 offer no support for running the Apache service as a specific user with network privileges. In fact, Windows 95/98 offers no security on the local machine, either. This is the simple reason that the Apache Software Foundation never endorses the use of Windows 95/98 as a public httpd server. These facilities exist only to assist the user in developing web content and learning the Apache server, and perhaps as a intranet server on a secured, private network.

Apache HTTP Server





Apache HTTP Server

Using Apache with Cygwin

This document explains how to install, configure and run Apache 1.3 under the [Cygwin](#) layer for Microsoft Windows. Cygwin is a POSIX.1 emulation layer for 32-bit Microsoft Windows operating systems.

The Apache Group does not guarantee that this software will work as documented, or even at all. If you find any bugs, please document them on our [bug reporting page](#).

Latest development news, pre-compiled distribution binaries, and third-party modules as DLLs, may be found at <http://apache.dev.wapme.net/>. Contributions are highly welcome (please see [TODO](#) list); please submit your code or suggestions to the bug report page, or join the dev@httpd.apache.org mailing list.

The [Win32 port](#) of Apache is built on its own, custom code within Apache to assure interoperability with Windows operating systems. While it is considered release quality, it is slower and less thoroughly tested than the Unix ports. The Cygwin alternative uses the well tested Unix code by using the Cygwin portability layer for POSIX.1 emulation. The Cygwin port may suffer from gaps in security or reliability due to the interaction of the Cygwin compatibility layer with the native Windows API.

The [Win32 port](#) will be more familiar to most Windows users. The Cygwin port (including the build environment) will be more familiar to Unix admins and developers. Due to these two different code bases, the security and reliability of the two ports are unrelated to each other. The Win32 port should be considered the more secure of the two at this time. The Win32 port is recommended for most Windows users, however the Cygwin port offers an extra layer of compatibility for Unix developers.

Apache still performs best, and is most reliable, on Unix platforms. First benchmarks have shown that the same Apache setup on Cygwin performs about 30% slower than the corresponding native Windows version.

Most of this document assumes that you have a working Cygwin installation and want to compile Apache yourself from the original distribution sources.

-
- [History of Apache for Cygwin](#)
 - [Differences from Apache for Windows \(native\)](#)
 - [Requirements](#)
 - [Downloading Apache for Cygwin](#)
 - [Configuring and Installing Apache for Cygwin](#)
 - [Running Apache for Cygwin](#)
 - [Running Apache for Cygwin as a Service](#)
-

History of Apache for Cygwin

Cygwin support for Apache began with Apache 1.2.6 and Cygwin b18.

Due to licensing issues there has not been an official binary distribution until Red Hat Inc. (who merged with Cygnus Solutions Inc.) changed their Cygwin license to ensure compiled executables do not fall under GPL license if the distributed software is considered as open source.

Cygwin is supported in the official source distributions from Apache 1.3.20 and later. Pre-compiled binaries for the Cygwin

platform (without the `cygwin1.dll`) will be supplied at <http://www.apache.org/httpd> for each released version.

Differences from Apache for Windows (native)

Both versions, Apache for Windows and Apache for Cygwin, are designed to run on the same operating systems, the Windows NT and Windows 2000 family. But there are considerable differences between the two flavors.

While Apache for Windows is a native Windows port, Apache for Cygwin relies on the Cygwin POSIX.1 emulation layer provided by the `cygwin1.dll` dynamic library to create a Unix compatible environment. Therefore we consider Apache for Cygwin closer to the Unix side than to the Windows side, even while it runs on Windows.

Most significant differences are the amount of changes to the source code needed to compile and run Apache on the Cygwin platform. While the native Windows port needs major changes and platform specific additions, the Cygwin based port changes are very small and most of the Unix source code can be used without major changes on the Cygwin platform.

When to use Apache for Cygwin and/or Apache versus Windows?

Apache for Cygwin is intended to be most useful if you want a seamless transition from Unix systems to Windows systems for your HTTP services.

If you are using Windows NT or Windows 2000 for development and office purposes, but your productive HTTP server environments are Unix based systems, you may use Apache for Cygwin to develop on Windows and simply copy whole Apache configurations (*i.e.*, `httpd.conf`) and Perl (`mod_perl`), PHP (`mod_php`) or Python (`mod_snake`) applications to your productive Unix systems.

What about modules (`mod_*`) for Apache for Cygwin?

Apache for Cygwin can be built with most of the available Apache modules with no or minimal changes. Many popular modules have been compiled and tested with Apache for Cygwin, including `mod_dav`, `mod_ssl`, `mod_php`, `mod_perl`, `mod_gzip`, and `mod_jserv`.

While there are developers who directly support the Windows native port of Apache, very few module developers do. That is why it is can be difficult to make a Unix-based Apache installation with third-party modules work the same way on the Windows side using the native port. Apache for Cygwin makes this much easier.

What are the differences in the configuration files?

While the Apache for Windows port uses Windows native path names to refer files and directories, like

```
# httpd.conf (for Windows)
DocumentRoot "c:/apache/htdocs"
```

Apache for Cygwin can use unmodified POSIX style path names like

```
# httpd.conf (for Cygwin)
DocumentRoot "/usr/local/apache/htdocs"
```

What about performance?

Apache for Cygwin is not as high-performance as Apache for Windows on the same hardware.

This is to be expected, because Cygwin emulates a Unix environment on a "foreign" operating system, while Apache for Windows uses Windows code in its own native environment. First benchmark results have shown that Apache for Cygwin is about 30% slower than native Apache for Windows counterpart.

Requirements

This Apache 1.3 port for Cygwin is designed to run on Windows NT 4.0 and Windows 2000, **NOT** on Windows 95 or 98. Windows NT 4.0 and Windows 2000 have both been successfully tested and approved. In all cases TCP/IP networking must be installed.

Cygwin 1.x is required to compile and run this version. Cygwin 1.1.8 and 1.3.2 have been tested and approved on both supported OSes.

Note: If you want to compile shared DLL modules using [apxs](#) you will need a patched version of `ld.exe` with

--auto-import support, which can be found at <http://sourceforge.net/projects/kde-cygwin/>.

Downloading Apache for Cygwin

The Cygwin platform is supported out of the box by Apache 1.3.20 and later. This means there is no extra download required for the Cygwin platform. The latest version of Apache can be found on the Apache httpd web site at <http://httpd.apache.org/>. The site lists the current release, any more recent development versions, and information on any mirror sites.

Configuring and Installing Apache for Cygwin

Apache on Cygwin is configured and compiled the same way as on Unix systems. Refer to the general [configuration](#) and [installation](#) documents for details.

There are three ways to configure and build Apache for Cygwin, depending on how additional Apache modules should be used:

- **Static linked version**

To build a static linked version of httpd including additional modules, use the following commands in the shell:

```
$ cd apache_1.3.x
$ ./configure [--enable-module=module | --add-module=/path/to/module]
$ make
```

This will produce the required extra libraries or object files for *module* and link everything to `src/httpd.exe`.

- **Shared core, DLL linked version ('one-for-all' version)**

To build a DLL version of httpd including additional modules, use the following commands:

```
$ cd apache_1.3.x
$ ./configure --enable-rule=SHARED_CORE \
  [--enable-module=module | --add-module=/path/to/module]
$ make
$ make install
```

This will produce the required extra libraries or object files which hold all static linked code. Then `dllwrap` and `dlltool` will export all of those (including any additional module code) to the shared `libhttpd.dll` and create the `libhttpd.a` import library which is required for linking `httpd.exe`.

Note: After `make install` is performed you will find the resulting core DLL module `libhttpd.dll` within `/usr/local/apache/libexec`. This is due to the installation process. Please move the file to Apache's bin directory, i.e.

```
$ mv /usr/local/apache/libexec/libhttpd.dll /usr/local/apache/bin
```

The core DLL module `libhttpd.dll` is the *only* file that should reside in `/usr/local/apache/bin` directory. All other shared DLL modules `mod_foo.dll` should be located in `/usr/local/apache/libexec`.

- **Shared DLL modules linked version**

This method is currently **ONLY** supported using a patched version of `ld.exe` which support the `--auto-import` option. Please see the [requirements](#) section for where to get that version.

To build a dynamic loadable DLL version of httpd which can load DLL modules on the fly (at runtime), proceed as follows:

- First build Apache's shared core as follows:

```
$ cd apache_1.3.x
$ ./configure --enable-rule=SHARED_CORE --enable-module=so \
```

```

    [--enable-module=module | --add-module=/path/to/module] \
    [--enable-shared=module]
$ make

```

You will notice that there is a warning message shown which lets you know that the shared core DLL library `src/libhttpd.dll` is missing while trying to link the shared DLL modules `mod_foo.dll`.

Unfortunately while Apache's build process the shared modules are linked before the shared core import library `src/libhttpd.dll` has been made. The shared modules depend on this import library, so they can not link for the first time you run `make`.

- Re-run `make` to build the shared module DLLs and install the whole package to the installation directory:

```

$ make
$ make install

```

All shared modules are placed into `libexec`, including the shared core DLL `libhttpd.dll`. When Apache's `/bin/httpd` is started it has to dynamically link `libhttpd.dll` while runtime. That is why you have to place the shared core DLL `libhttpd.dll` to the same directory where `httpd.exe` resides, i.e. `/usr/local/apache/bin`.

- Add configuration directives to `conf/httpd.conf` to load and activate shared DLL modules at runtime:

```

# httpd.conf
[...]
LoadModule foo_module    libexec/mod_foo.dll
AddModule mod_foo.c
[...]

```

● Using `apxs` to create shared DLL modules

To make the extending `httpd` with shared DLL modules easier, you can use [apxs](#).

Make sure you have configured `$CFG_LDFLAGS_SHLIB` within `apxs` to include the `--shared` directive and the path to the shared code DLL `libhttpd.dll`.

After performing `make install` you will probably have the following lines within your `apxs`:

```

# apxs
[...]
my $CFG_LD_SHLIB      = q(dllwrap --export-all --output-def libhttpd.def --implib
libhttpd.a --driver-name gcc);          # substituted via Makefile.tmpl
my $CFG_LDFLAGS_SHLIB = q(-g); # substituted via Makefile.tmpl
my $CFG_LIBS_SHLIB    = q();          # substituted via Makefile.tmpl
[...]

```

Change these to reflect the new compile options needed for shared DLL modules as follows:

```

# apxs
[...]
my $CFG_LD_SHLIB      = q(gcc);          # substituted via Makefile.tmpl
my $CFG_LDFLAGS_SHLIB = q(-g --shared); # substituted via Makefile.tmpl
my $CFG_LIBS_SHLIB    = q(/path/to/libhttpd.dll);          # substituted via
Makefile.tmpl
[...]

```

Now you should be able to create a shared DLL module from a `mod_foo.c` source file with:

```
$ apxs -c mod_foo.c -o mod_foo.dll
```

Place the resulting DLL in Apache's `libexec` directory, so the `dlopen()` function within the compiled in `mod_so.c` module can find and load it at runtime.

Running Apache for Cygwin

Apache on Cygwin can be started and stopped in the same manner as on Unix systems. You may also use the [apachectl](#) tool for starting and stopping Apache.

- **Starting Apache**

If installed with the default Apache directory layout, you can start `httpd` as follows:

```
$ /usr/local/apache/bin/httpd
```

An explicit background indicator (&) is not required. The parent process is automatically detached from the current terminal. Check the global `error_log` to see if Apache started cleanly without any major problems.

- **Stopping Apache**

To stop Apache send at least a `SIGTERM` signal to the parent `httpd` process:

```
$ kill -TERM `cat /usr/local/apache/logs/httpd.pid`
```

- **Gracefully Restarting Apache**

In order to update configuration directives and reload the `httpd.conf` configuration file, send a `SIGHUP` to the parent `httpd` process:

```
$ kill -HUP `cat /usr/local/apache/logs/httpd.pid`
```

Running Apache for Cygwin as a Service

Apache on Cygwin can be invoked as a Windows NT or Windows 2000 service. Cygwin has its own `cygrunsrv.exe` facility to define, remove, start, and stop services as follows:

- **Installing Apache as a new Service**

Use the following statement to install `httpd.exe` as a new service:

```
$ cygrunsrv -I service_name -p /usr/local/apache/bin/httpd.exe [-a arguments] \
  [-e VAR=VALUE] [-t auto|manual] [-u user] [-w passwd]
```

Where `-a` is used to pass command line arguments (such as `-DFOO` defines) to `httpd.exe`, and `-e` is used to pass environment variables. If necessary you may use the `-t` options to set the autostart configuration for the service. If you want the new service to run under a different userid, you will have to supply the `-u` and `-w` options.

- **Starting Apache as a Service**

After the new service is installed it can be started using the following command:

```
$ cygrunsrv -S service_name
```

Check your process table and global `error_log` file to ensure Apache has started without any major problems.

- **Stopping an Apache Service**

A running Apache service may be stopped using the following command:

```
$ cygrunsrv -E service_name
```

This will stop all running `httpd.exe` processes and shutdown the HTTP service for the machine.

- **Removing an Apache Service**

An installed Apache service may be removed from Windows NT or Windows 2000 using the following command:

```
$ cygrunsrv -R service_name
```

This will remove your previously defined and installed service from the machine.

Please refer to the man page for `cygrunsrv` and the Cygwin mailing list for further details of how services are invoked.

Any additional contributions to this document and the Cygwin support for Apache are highly welcome. Please send them to Stipe Tolj <tolj@wapme-systems.de>.

Apache HTTP Server





Apache HTTP Server

Overview of the Apache EBCDIC Port

As of Version 1.3, the Apache HTTP Server includes a port to (non-ASCII) mainframe machines which use the EBCDIC character set as their native codeset.

(Initially, that support covered only the Fujitsu-Siemens family of mainframes running the [BS2000/OSD operating system](#), a mainframe OS which features a SVR4-derived POSIX subsystem. Later, the two IBM mainframe operating systems TPF and OS/390 were added).

EBCDIC-related conversion functions

The EBCDIC related directives [EBCDICConvert](#), [EBCDICConvertByType](#), and [EBCDICKludge](#) are available **only if the platform's character set is EBCDIC** (This is currently only the case on Fujitsu-Siemens' BS2000/OSD and IBM's OS/390 and TPF operating systems). EBCDIC stands for *Extended Binary-Coded-Decimal Interchange Code* and is the codeset used on mainframe machines, in contrast to ASCII which is ubiquitous on almost all micro computers today. ASCII (or its extension *latin1*) is the basis for the HTTP transfer protocol, therefore all EBCDIC-based platforms need a way to configure the code set conversion rules required between the EBCDIC based mainframe host and the HTTP socket protocol.

On an EBCDIC based system, HTML files and other text files are usually saved encoded in the native EBCDIC code set, while image files and other binary data are stored with identical encoding as on ASCII based machines. When the Apache server accesses documents, it must therefore make a distinction between text files (to be converted to/from ASCII, depending on the transfer direction) and binary files (to be delivered unconverted). Such a distinction can be made based on the assigned MIME type, or based on the file extension (*i.e.*, files sharing a common file suffix).

By default, the configuration is symmetric for input and output (*i.e.*, when a PUT request is executed for a document which was returned by a previous GET request, then the resulting uploaded copy should be identical to the original file). However, the conversion directives allow for specifying different conversions for input and output.

The directives [EBCDICConvert](#) and [EBCDICConvertByType](#) are used to assign the conversion setting (On or Off) based on file extensions or MIME types. Each configuration setting can be defined for input only (*e.g.*, PUT method), output only (*e.g.*, GET method), or both input and output. By default, the conversion setting is applied for input and output.

Note that after modifying the conversion settings for a group of files, it is not sufficient to restart the server. The reason for this is the fact that a cached copy of a document (in a browser or proxy cache) will not get revalidated by contents, but only by date. Since the modification time of the document did not change, browsers will assume they can reuse the cached copy. To recover from this situation, you must either clear all cached copies (browser and proxy cache!), or update the modification time of the documents (using the `touch` command on the server).

Note also that server-parsed documents (CGI scripts, .shtml files, and other interpreted files like PHP scripts etc.) are not subject to any input conversion and must therefore be stored in EBCDIC form on the server side.

In absense of any [EBCDICConvertByType](#) directive, and if no matching [EBCDICConvert](#) was found, Apache falls back to an internal heuristic which assumes that all documents with MIME types starting with "text/", "message/" or "multipart/" as well as the MIME type "application/x-www-form-urlencoded" are text documents stored in EBCDIC, whereas all other documents are binary files.

In order to provide backward compatibility with older versions of apache, the [EBCDICKludge](#) directive allows for a less powerful mechanism to control the conversion of documents to and from EBCDIC.

Note:

The `EBCDICKludge` directive is deprecated, since its functionality is superseded by the more powerful [EBCDICConvert](#) and [EBCDICConvertByType](#) directives.

The directives are applied in the following order:

1. First, the configured [EBCDICConvert](#) directives in the current context are evaluated in configuration file order. As soon as a matching file extension is found, the search stops and the configured conversion is applied. `EBCDICConvert` settings inherited from parent directories are tested after the more specific (deeper) directory levels.
2. If the `EBCDICKludge` is in effect, the next step tests for a MIME type of the format `type/x-ascii-subtype`. If the document has such a type, then the "x-ascii-" substring is removed and the conversion set to Off.
3. In the next step, the configured [EBCDICConvertByType](#) directives are evaluated in configuration file order. If the document has a matching MIME type, the search stops and the configured conversion is applied. `EBCDICConvertByType` settings inherited from parent directories are tested after the more specific (deeper) directory levels. If no [EBCDICConvertByType](#) directive at all exists in the current context, the server falls back to the simple heuristics which assume that MIME types starting with "text/", "message/" or "multipart/" (plus the special type "application/x-www-form-urlencoded" used in simple POST requests) imply a conversion, while all the rest is delivered unconverted (*i.e.*, binary).

Technical Details

Since all Apache input and output is based upon the `BUFF` data type and its methods, the easiest solution was to add the actual conversion to the `BUFF` handling routines. The conversion must be settable at any time, so `BUFF` flags were added which define whether a `BUFF` object has currently enabled conversion or not. Two such flags exist: one for data read from the client (ASCII to EBCDIC conversion) and one for data returned to the client (EBCDIC to ASCII conversion).

During sending of the header, Apache determines (based on the returned MIME type for the request) whether conversion should be used or the document returned unconverted. It uses this decision to initialize the `BUFF` flag when the response output begins. Modules should therefore determine the MIME type for the current request before initiating the response by calling `ap_send_http_headers()`.

The `BUFF` flag is modified at several points in the HTTP protocol:

- **set** (In and Out) before a request is received (because the request and the request header lines are always in ASCII format)
- **set/unset** (for Input data) when the request body is received - depending on the content type of the request body (because the request body may contain ASCII text or a binary file)
- **set** (for returned Output) before a response header is sent (because the response header lines are always in ASCII format)
- **set/unset** (for returned Output) when the response body is sent - depending on the content type of the response body (because the response body may contain text or a binary file)

Additional transparent transitions may occur for extracting/inserting the HTTP/1.1 chunking information from/into the input/output body data stream, and for generating *multipart* headers for *range* requests. (See RFC2616 and `src/main/http_protocol.c` for details.)

Porting Notes

1. The relevant changes in the source are `#ifdef`d into two categories:

`#ifdef CHARSET_EBCDIC`

Code which is needed for any EBCDIC based machine. This includes character translations, differences in contiguity of the two character sets, flags which indicate which part of the HTTP protocol has to be converted and which part doesn't *etc.*

`#ifdef _OSD_POSIX | TPF | OS390`

Code which is needed for the Fujitsu-Siemens BS2000/OSD | IBM TPF | IBM OS390 mainframe platforms only. This deals with include file differences and socket and fork implementation topics which are only required on the respective platform.

2. The possibility to translate between ASCII and EBCDIC at the socket level (on BS2000 POSIX, there is a socket option which supports this) was intentionally *not* chosen, because the byte stream at the HTTP protocol level consists of a mixture of protocol related strings and non-protocol related raw file data. HTTP protocol strings are always encoded in ASCII (the GET request, any Header: lines, the chunking information *etc.*) whereas the file transfer parts (*i.e.*, GIF images, CGI output *etc.*) should usually be just "passed through" by the server. This separation between "protocol string" and "raw data" is reflected in the server code by functions like `bgets()` or `rvputs()` for strings, and functions like `bwrite()` for binary data. A global translation of everything would therefore be inadequate.
(In the case of text files of course, provisions must be made so that EBCDIC documents are always served in ASCII)
This port therefore features a built-in protocol level conversion for the server-internal strings (which the compiler translated to EBCDIC strings) and thus for all server-generated documents.
3. By examining the call hierarchy for the BUFF management routines, I added an "ebcdic/ascii conversion layer" which would be crossed on every `puts/write/get/gets`, and conversion flags which allowed enabling/disabling the conversions on-the-fly. Usually, a document crosses this layer twice from its origin source (a file or CGI output) to its destination (the requesting client): file -> Apache, and Apache -> client.
The server can now read the header lines of a CGI-script output in EBCDIC format, and then find out that the remainder of the script's output is in ASCII (like in the case of the output of a WWW Counter program: the document body contains a GIF image). All header processing is done in the native EBCDIC format; the server then determines, based on the type of document being served, whether the document body (except for the chunking information, of course) is in ASCII already or must be converted from EBCDIC.
4. By default, Apache assumes that documents with the MIME types "text/*", "message/*", "multipart/*" and "application/x-www-form-urlencoded" are text documents and are stored as EBCDIC files, whereas all other files are binary files (and stored in a byte-identical encoding as on an ASCII machine).
These defaults can be overridden on a [by-MIME-type](#) and/or [by-file-extension](#) basis, using the directives

```
EBCDICConvertByType {On|Off} [= {In|Out|InOut}] mimetype [...]  
EBCDICConvert {On|Off} [= {In|Out|InOut}] fileext [...]
```

where the *mimetype* argument may contain wildcards.

5. Before adding the flexible conversion, non-text documents were always served "binary" without conversion. This seemed to be the most sensible choice for, *e.g.*, GIF/ZIP/AU file types (It of course requires the user to copy them to the mainframe host using the "rcp -b" binary switch), but proved to be inadequate for MIME types like `model/vrml`, `application/postscript` and `application/x-javascript`.
6. Server parsed files are always assumed to be in native (*i.e.*, EBCDIC) format as used on the machine (because they do not cross the conversion layer when being read), and are converted after processing.
7. For CGI output, the CGI script determines whether a conversion is needed or not: by setting the appropriate Content-Type, text files can be converted, or GIF output can be passed through unmodified (depending on the conversion configured in the script's context).

Document Storage Notes

Binary Files

When exchanging binary files between the mainframe host and a Unix machine or Windows PC, be sure to use the ftp "binary" (TYPE I) command, or use the `rcp -b` command from the mainframe host (the `-b` switch is not supported in unix `rcp`'s).

Text Documents

The default assumption of the server is that Text Files (*i.e.*, all files whose Content-Type: starts with `text/`) are stored in the native character set of the host, EBCDIC.

Server Side Included Documents

SSI documents must currently be stored in EBCDIC only. No provision is made to convert them from ASCII before processing. The same holds for other interpreted languages, like mod_perl or mod_php.

Apache HTTP Server



Overview of the Apache TPF Port

[[Configuration Files](#) | [What's Available](#) | [CGI Scripts](#) | [Options](#) | [Porting Notes](#)]

This version of Apache includes changes allowing it to run on IBM's EBCDIC-based [TPF](#) (Transaction Processing Facility) operating system.

Unless otherwise noted TPF version 4.1 PUT09 is required.

Refer to [install-tpf.html](#) for step-by-step installation instructions.

This port builds upon the [EBCDIC changes](#) previously made to Apache.

Apache Configuration Files

The distributed configuration files (httpd.conf-dist and mime.types, both located in the conf subdirectory) work on TPF. Performance considerations may dictate setting KeepAlive to "Off" (the default is "On") or lowering the Timeout value from the default 300 seconds (5 minutes) in order to reduce the number of active ECBs on your system.

What's Available in this Version

The Apache organization provides [online documentation](#) describing the various modules and components of the server.

Components/modules tested on TPF:

- alloc.c
- ap_base64.c
- ap_checkpass.c
- ap_cpystn.c
- ap_ebcdic.c
- ap_fnmatch.c
- ap_md5c.c
- ap_sha1.c
- ap_signal.c
- ap_slack.c
- ap_snprintf.c
- buff.c
- buildmark.c
- gen_test.char.c
- gen_uri_delims.c
- htpasswd.c (*requires PUT10*)
- http_config.c
- http_core.c
- http_log.c
- http_main.c
- http_protocol.c
- http_request.c

The Apache TPF Port

- http_vhost.c
- logresolve.c (*requires PUT10*)
- mod_access.c (*Use of mod_access directives "allow from" & "deny from" with host names (verses ip addresses) requires PUT10*)
- mod_actions.c
- mod_alias.c
- mod_asis.c
- mod_auth.c
- mod_auth_anon.c
- mod_autoindex.c
- mod_cern_meta.c
- mod_cgi.c (*requires PUT10*)
- mod_digest.c
- mod_dir.c
- mod_env.c
- mod_example.c
- mod_expires.c
- mod_headers.c
- mod_imap.c
- mod_include.c (*CGI execution requires TPF version 4.1 PUT10*)
- mod_info.c
- mod_log_agent.c
- mod_log_config.c
- mod_log_referer.c
- mod_mime.c
- mod_mime_magic.c
- mod_negotiation.c
- [mod_put.c](#) (*third party module*)
- mod_proxy.c
- mod_setenvif.c
- mod_speling.c
- mod_status.c
- mod_tpf_shm_static.c (*third party module, requires PUT10*)
- mod_unique_id.c (*requires PUT10*)
- mod_userdir.c
- mod_usertrack.c
- os.c
- os-inline.c
- proxy_cache.c
- proxy_connect.c
- proxy_ftp.c
- proxy_http.c
- proxy_util.c
- regular expression parser

The Apache TPF Port

- regular expression test tool (*requires PUT10*)
- rfc1413.c
- rotatelog.c (*requires PUT10*)
- util.c
- util_date.c
- util_md5.c
- util_script.c
- util_uri.c

Components/modules not yet supported on TPF:

- htdigest.c
- lib/expat-lite
- mod_auth_digest.c
- mod_rewrite.c
- mod_vhost_alias.c

Components/modules that don't apply or that probably won't ever be available on TPF:

- ab.c
- ap_getpass.c
- mod_auth_db.c
- mod_auth_dbm.c
- mod_auth_db.module
- mod_mmap_static.c
- mod_so.c
- suexec.c

How to Use CGI Scripts

The following is a very simple example of a CGI script ("Hello World") and the necessary steps to run it. Refer to the [mod_cgi module](#) for additional information.

Add necessary directives to httpd.conf:

Example:

```
ScriptLog logs/script_log
ScriptAlias /cgi-bin/ /usr/local/apache/cgi-bin/
```

A request for `http://myserver/cgi-bin/filename.cgi` would cause the server to run the script `/usr/local/apache/cgi-bin/filename.cgi`

Create the CGI script:

For this example QZZ1 is the name of the TPF program that will be executed by the CGI script. The directory path must match what is in the httpd.conf file for ScriptAlias directive.

```
zfile echo "#!QZZ1" > /usr/local/apache/cgi-bin/filename.cgi
zfile cat /usr/local/apache/cgi-bin/filename.cgi      (this should display #!QZZ1)
```

Mark the script as executable:

```
zfile chmod 755 /usr/local/apache/cgi-bin/filename.cgi
```

Create, load, and activate a loadset containing the CGI program (QZZ1xx):

```
/** QZZ1-- simple "Hello world" program to demonstrate basic CGI output **/  
  
#include <stdio.h>  
  
void main() {  
  
/** Print the CGI response header, required for all HTML output. **/  
/** Note the extra \n, to send the blank line. **/  
  
printf("Content-type: text/html\n\n");  
  
/** Print the HTML response page to stdout. **/  
printf("<html>\n");  
printf("<head><title> CGI Output </title></head>\n");  
printf("<body>\n");  
printf("<h1> Hello world </h1> \n");  
printf("</body>\n");  
printf("</html>\n");  
  
}
```

Request the CGI script from a browser:

```
http://myserver/cgi-bin/filename.cgi
```

How to Use Apache Command Line Options

You cannot run Apache from the command line on TPF. However you can use those Apache command line options which don't actually start the server. This requires PJ27277 which shipped on PUT13.

Supported Apache options:

-d *directory*

Specify an alternate initial ServerRoot *directory*. Default is /usr/local/apache.

-f *file*

Specify an alternate server configuration *file*. Default is conf/httpd.conf.

-h

List a short summary of available command line options. (Note that this outputs all options, not just those supported on TPF.)

-l

List modules compiled into the server.

-L

List available configuration directives. (Note that this outputs all configuration directives, not just those supported on TPF.)

-S

Show the settings as parsed from the configuration file. Currently only shows the virtualhost settings.

-t
Run syntax tests for configuration files (with document root checks)

-T
Run syntax tests for configuration files (without document root checks)

-v
Show the version number.

-V
Show the version number and various compile settings.

See <http://httpd.apache.org/docs/programs/httpd.html> for more information about these command line options.

Note: On TPF Apache arguments are supported only on the command line, not through the ZINET XPARAM field.

Setup

Ensure Apache (CHTA) is loaded

Create the httpd script:

```
zfile echo "#!CHTA" > /bin/httpd
zfile cat /bin/httpd (this should display #!CHTA)
```

Mark the script as executable:

```
zfile chmod 755 /bin/httpd
```

(See "ZFILE-Activate a TPF Segment or Script" in the *Operations* guide for more information:
<http://www.ibm.com/tpf/pubs/tpfpubs.htm>.)

Example 1

```
zfile httpd -v
```

```
FILE0001I 11.43.09 START OF DISPLAY FROM httpd -v
Server version: Apache/1.3.20 (TPF)
Server built: May 23 2001 09:39:22
END OF DISPLAY
```

Example 2

```
zfile httpd -t -f /usr/local/apache/conf/httpd.conf.new
```

```
FILE0002I 11.47.26 START OF ERROR DISPLAY FROM httpd -t ...
Syntax OK
END OF DISPLAY
```

Porting Notes

Changes made due to differences between UNIX and TPF's process models:

- **Signals:** On TPF a signal that is sent to a process remains unhandled until the process explicitly requests that signals be handled using the `tpf_process_signals()` function. Additionally, the default action for an alarm on TPF is to take an OPR-7777 dump and exit. (On UNIX the default is the equivalent of `exit()` with no dump taken.) These differences necessitated a few modifications:
 - - bypass the use of `ap_block_alarms()` & `ap_unblock_alarms()`
 - add `tpf_process_signals()` calls
 - add `select()` calls to prevent blocking.

Find that function...

Some simple functions & definitions initially needed to be added on TPF, such as `FD_SET()`. We've put these in `src/os/tpf/os.h` for now.

EBCDIC changes:

TPF-specific conversion tables between US-ASCII and EBCDIC (character set IBM-1047 to be exact) were created.

Miscellaneous, minor changes:

Various minor changes (such as casting) were made due to differences in how some functions are implemented on TPF.

[[top](#) | [Configuration Files](#) | [What's Available](#) | [CGI Scripts](#) | [Options](#) | [Porting Notes](#)]

Installing the Apache 1.3 HTTP Server on TPF

[[Download](#) | [Compilation](#) | [Installation](#) | [VisualAge](#)]

This document outlines the steps needed to install Apache onto a TPF system.

You should first read [readme-tpf.html](#) for basic information on the port of Apache to TPF including required PUT level and supported modules.

Download

Releases of the Apache server are compressed into a "tarball" file which must be downloaded to your PC. Additionally the source code from the tarball will need to be copied onto an OS/390 UNIX System Services machine (later referred to simply as "OS/390 UNIX") for compiling. Here are all the details on how to get Apache and how to get it where it needs to be:

1. Download the compressed Apache files (the "tarball") to your PC. The file name on the web site will be something like `apache_1.3.xx.tar.Z`.

TIP: Be sure to keep the `.tar.Z` extension when choosing the name of the PC file.

2. Decompress the tarball on your PC using WinZip or some other PC decompression tool.

TIP: If you are using WinZip verify that the "*TAR File Smart CR/LF Conversion*" option (under Options, Configuration) is NOT checked.

This is what you can expect if you use WinZip:

- open the tarball with WinZip (this can usually be done simply by double-clicking on the downloaded tarball)
- you will be told that the archive contains one file (such as `apache_1.3.xx.tar`) - allow WinZip to decompress it to a temporary folder
- extract the archived files onto your PC - you'll be using files from the `conf`, `htdocs`, and `icons` directories later in the install phase

3. FTP the tarball to your OS/390 UNIX machine using binary mode:

4. ○ activate FTP in an MSDOS window: `ftp your.os390.unix.machine.com`

○ sign in

○ set mode to binary: `binary`

○ send the file to OS/390 UNIX:

```
send c:\downloaded_filename.tar.Z os390_unix_filename.tar.Z
```

○ exit FTP: `bye`

TIP: UNIX file names are case sensitive. If you use an NFS client to transfer files from your PC to OS/390 UNIX (instead of using FTP as described above) verify that the NFS drive will transfer the file names with upper/lower case preserved.

5. Decompress the tarball on OS/390 UNIX: `gunzip os390_unix_filename.tar.Z`

Note that the `.tar.Z` file will be replaced by the decompressed `.tar` archive file.

6. Extract the archived files necessary for compiling Apache:

```
pax -rvkf os390_unix_filename.tar -o from=ISO8859-1,to=IBM-1047 "*/src"
```

7. Remove unnecessary subdirectories:

- `cd apache_1.3.xx/src/os`
- `rm -r bs2000 cygwin mpeix netware os2 os390 win32`
-

Compilation

Apache supports the notion of "optional modules". However, the server has to know which modules are compiled into it. In order for those modules to be effective, it is necessary to generate a short bit of code (`modules.c`) which simply has a list of them. If you are using the `Configure` utility and `make`, `modules.c` and other necessary files will be created for you automatically.

The provided instructions assume a c89 compiler and have been tested on an OS/390 UNIX machine running at version 2.6 that contained both OS/390 UNIX and TPF C header files. If you are using a platform other than OS/390 UNIX you may need to modify `src/os/tpf/TPFExport` and `src/Configure` to match your environment.

TIP: Editing files on your PC prior to moving them to OS/390 UNIX may result in the loss/addition of unprintable characters. Files of concern include shell scripts and `src/Configuration`. The most common problems are with tab characters and CR/LF characters. Most editors will handle the CR/LF problem correctly but none seem to handle tab characters. If you need to edit files prior to moving them to OS/390 UNIX, edit them in a UNIX editor such as `vi` or `emacs`.

Note that OS/390 UNIX commands in this section are shown in **bold**, are case sensitive, and must be made from the "src" directory.

1. Switch to the source code subdirectory: **cd apache_1.3.xx/src**
2. Overlay `src/Configuration` with `src/Configuration.tmpl`: **cp Configuration.tmpl Configuration**
3. Edit `src/Configuration`. It contains the list and settings of various "Rules" and an additional section at the bottom that determines which modules to compile:
 4. ○ Adjust the Rules and `EXTRA_CFLAGS | LIBS | LDFLAGS | INCLUDES` if you feel so inclined.
 - Comment out (by preceding the line with a "#") lines corresponding to those modules you DO NOT wish to include.
 - Uncomment (by removing the initial "#", if present) lines corresponding to those modules you wish to include or add new lines corresponding to any custom modules you have written. The [readme-tpf.html](#) document lists the modules that have been tested on TPF.

The modules placed in the Apache distribution are the ones that have been tested and are used regularly by various members of the Apache development group. Additional modules contributed by members or third parties with specific needs or functions are available at <http://modules.apache.org/>. There are instructions on that page for linking these modules into the core Apache code.

5. Indicate whether the `non_socket_select` function is implemented on your system.

If you are on a PUT12 or higher system, or have PJ26895 installed, then you probably support `non_socket_select`. You can verify this by looking for the `non_socket_select` prototype in your system header files (specifically `i$pwbl.h`).

If your TPF system supports `non_socket_select` do *one* of the following:

- add `#define TPF_HAVE_NONSOCKET_SELECT` to `src/os/tpf/os.h` *or*
- add `-DTPF_HAVE_NONSOCKET_SELECT` to the `_C89_OPTIONS` export in `src/os/tpf/TPFExport`

Otherwise:

- add "#define TPF_NO_NONSOCKET_SELECT" to `src/os/tpf/os.h` *or*
- add "-DTPF_NO_NONSOCKET_SELECT" to the `_C89_OPTIONS` export in `src/os/tpf/TPFExport`

Without `non_socket_select` CGI output is buffered and only sent to the browser when the CGI program finishes.

6. Indicate whether the `tpf_sawnc` function is implemented on your system.

If you are on a PUT10 or higher system, or have PJ27387/PJ26188 installed, then you probably support `tpf_sawnc`. You can verify this by looking for the `tpf_sawnc` prototype in your system header files (either `tpfapi.h` or `i$fsdd.h`).

If your TPF system supports `tpf_sawnc` do *one* of the following:

- add "#define TPF_HAVE_SAWNC" to `src/os/tpf/os.h` *or*
- add "-DTPF_HAVE_SAWNC" to the `_C89_OPTIONS` export in `src/os/tpf/TPFExport`

Otherwise:

- add "#define TPF_NO_SAWNC" to `src/os/tpf/os.h` *or*
- add "-DTPF_NO_SAWNC" to the `_C89_OPTIONS` export in `src/os/tpf/TPFExport`

7. The use of `tpf_sawnc` allows for a cleaner shutdown of Apache.

8. Set the TPF environment variables: `. os/tpf/TPFExport`

TIP: The initial period and blank on the command are required to ensure the environment variables exist beyond the scope of the shell script.

This script will set the environment variables required to compile the programs for TPF. Verify that the export variables are valid for your installation, in particular, the system include file directories. The system include files must reside on your OS/390 UNIX system in the appropriate file structure similar to `/usr/include` and `/usr/include/sys`. DO NOT modify the `TPF=YES` export variable. If this is changed, the "Configure" script will not recognize TPF.

9. Run the "Configure" script: **Configure**

This generates `modules.c`, `include/ap_config_auto.h`, and necessary Makefiles:

```
Using config file: Configuration
Creating Makefile
+ configured for TPF platform
+ setting C compiler to c89
+ setting C pre-processor to c89 -E
+ checking for system header files
+ adding selected modules
+ checking sizeof various data types
Creating Makefile in support
Creating Makefile in regex
Creating Makefile in os/tpf
Creating Makefile in ap
Creating Makefile in main
Creating Makefile in lib/expat-lite
Creating Makefile in modules/standard
$ _
```

If you want to maintain multiple configurations, you can say, *for example*,

Configure -file Configuration.2nd

```
Using config file: Configuration.2nd
Creating Makefile
+ configured for <whatever> platform
+ setting C compiler to <whatever>
et cetera
```

If you receive an error such as "Configure 146: FSUM7351 not found" the most likely explanation is that one or more of the make related files were edited on a non-UNIX platform, corrupting the end-of-line marks. Verify that lines ending with "\" in the flagged file do not have trailing spaces. Using the vi editor and the sample error above as an example...

```
pull up the flagged file:      vi Configure
turn on punctuation:         :set list
go to the line in question:   146G
    or find a line with a "\":  /\
```

The end of line should display as "\\$". If it is displayed as "\ \$" (with a blank between \ and \$) then you should revert to the distributed version of the file and make the site-specific changes again using a UNIX compatible editor such as vi or emacs. Then try the Configure command again.

```
close the file:              :q (or :quit!)
```

10. Edit include/ap_config.h if you do *not* want the scoreboard kept in shared memory.

The default behavior for Apache on all platforms *except* TPF is to use the file system for maintaining the scoreboard (which holds current Apache children status). The default behavior for Apache on TPF is to use shared memory. This reduces file activity for the parent Apache ECB and improves performance. If you are on a pre-PUT10 system you must change ap_config.h to use either system heap or the file system.

To use system heap for the scoreboard replace #define USE_SHMGET_SCOREBOARD with #define USE_TPF_SCOREBOARD in the TPF section of ap_config.h.

If you prefer instead to use the file system, remove both #define USE_SHMGET_SCOREBOARD and #define USE_TPF_SCOREBOARD from the TPF section of ap_config.h

The change will only take effect after Apache is (re)compiled.

11. Now compile the programs: **make**

Besides compiling, make also runs src/main/gen_test_char.c and src/main/gen_uri_delims.c in order to create src/main/test_char.h and src/main/uri_delims.h respectively

- The following compilation warning is expected and can be ignored:

```
util_uri.c: Function argument assignment between types "unsigned char*" and
"const unsigned char*" is not allowed.
```

- If during compilation you get a warning about a missing 'regex.h', set WANTHSREGEX=yes in the src/Configuration file and start back at the [Configure](#) step.
- If you get a 'Duplicate type specifier "long" ignored' error, add "-W 0,langlvl(extended)" to the _C89_OPTIONS export in src/os/tpf/TPFExport and start back at the [export](#) step

Installation

1. Link the compiled object files into a DLL. Sample link JCL has been included as `src/os/tpf/samples/linkhttp.jcl`. You will need to modify this JCL:
 - Change the IDs, data set names, and libraries for your particular site.
 - Add/remove `mod_XXX.o` files so they correspond to the `mod_XXX.o` lines in your `src/Configuration` file.

TIP: Do NOT include `gen_test_char.o` or `gen_uri_delims.o` in the link JCL since these files are only used during the make step.

2. Create a loadset. Sample loadset JCL has been included as `src/os/tpf/samples/loadset.jcl`. You will need to modify this JCL for your particular site.

A JCL condition code of 4 is expected since the C load module will contain no link map data.

3. Load (`ZOLDR LOAD`) and activate (`ZOLDR ACT`) the loadset on your test system.
4. Ensure that the program name you are using for Apache has `RESTRICT` and `KEY0` authorization. `zdpat chta (c-c)` will display allocation information. You can use `zapat chta restrict key0 (c-c)` to alter the authorization. Note that if the program name is unallocated, you must have the loadset for it activated or you will receive `INVALID PROGRAM NAME` from the `zdpat/zapat` entries.

5. Create the Apache run-time configuration file. The server requires a configuration file to initialize itself during activation. (Previously three configuration files were used.) Copy the distribution version, `/conf/httpd.conf-dist`, to `/conf/httpd.conf` and then edit the `/conf/httpd.conf` copy with your site specific information.

At a minimum you must change every occurrence of "`@@ServerRoot@@`" to your document server root (for example `/usr/local/apache`)

6. General documentation for Apache is located at <http://httpd.apache.org/docs/> and in the HTML pages included with the distribution (tarball) under the `/htdocs/manual` directory.
7. On TPF activate ZCLAW

Refer to the TCP/IP Offload Support section of the *TPF TCP/IP* publication for more information:
<http://www.ibm.com/tpf/pubs/tpfpubs.htm>.

Note: Apache does not currently work with Native Stack.

8. Using either TFTP or FTP, transfer the configuration file, icons, and web pages to your TPF system. A typical directory structure for Apache is as follows:

```
/usr/local/apache/conf
  /usr/local/apache/logs
  /usr/local/apache/icons
  /usr/local/apache/htdocs
```

All gif, jpg, and zip files should be transferred as binary; the configuration file and html pages should be transferred as text.

The logs directory must exist in order to avoid an `fopen` error while running Apache:

If you're running a PUT10 or higher version of TPF make the directory using the `zfile mkdir /usr/local/apache/logs` functional entry.

If you're running TPF version PUT09 TFTP an empty file into the logs subdirectory to create it.

Make sure Apache can write into the logs subdirectory by doing a `zfile chmod` on it with the appropriate permission settings.

Refer to the TFTP and FTP sections of the *TPF TCP/IP* publication for more information:
<http://www.ibm.com/tpf/pubs/tpfpubs.htm>.

9. On TPF add Apache to the Internet Daemon's tables using ZINET entries, the common case:

- For PUT11 and later use the DAEMON model for
Apache:**ZINET ADD S-APACHE PGM-*chta* MODEL-DAEMON USER-root**
- On pre-PUT11 systems use the NOLISTEN model
instead:**ZINET ADD S-APACHE PGM-*chta* MODEL-NOLISTEN**

TIP: Logic changes implemented with PUT11 cause ZINET to *not* restart NOLISTEN servers after ZOLDR ACT and ZOLDR DEACT entries. This means that Apache running as NOLISTEN on a PUT11 or later system will exit whenever any ZOLDR ACT or ZOLDR DEACT entry is made. Therefore at PUT11 you should switch to the DAEMON model and ensure that you have APARs PJ25761 and PJ27363 applied.

Refer to the Internet Daemon section of the *TPF TCP/IP* publication for more information:
<http://www.ibm.com/tpf/pubs/tpfpubs.htm>.

10. Start the server using the **ZINET START S-APACHE** command.

11. Request a page from your browser: **http://xx.xx.xx.xx** (where xx.xx.xx.xx is your IP address)

Compiling with VisualAge TPF

It is not required that make be used to compile Apache for TPF: Individual programs may be compiled using IBM's VisualAge TPF product. This is particularly useful when compiling selected programs for the Debug Tool.

The following VisualAge compile settings are required:

- "*DEFINE - Define preprocessor macro name(s)*" must include **TPF**, **CHARSET_EBCDIC**, **_POSIX_SOURCE**, and **USE_HSREGEX**
- "*LSEARCH - Path for user include files*" must include **../src/include** and **../src/os/tpf**
- "*DLL - Generate DLL code*" must be checked
- "*LONGNAME - Support long names*" must be checked

[[top](#) | [Download](#) | [Compilation](#) | [Installation](#) | [VisualAge](#)]



Apache HTTP Server

Using Apache With HP MPE/iX

This document explains how to compile, install, configure and run Apache 1.3 under HP MPE/iX.

The bug reporting page and new-httpd mailing list are NOT provided to answer questions about configuration or running Apache. Before you submit a bug report or request, first consult this document, the [Frequently Asked Questions](#) page and the other relevant documentation topics. If you still have a question or problem, post it to the [comp.sys.hp.mpe](#) newsgroup or the associated [HP3000-L mailing list](#), where many Apache users and several contributors are more than willing to answer new and obscure questions about using Apache on MPE/iX.

[groups.google.com's newsgroup archive](#) offers easy browsing of previous questions. Searching the newsgroup archives, you will usually find your question was already asked and answered by other users!

-
- [Requirements](#)
 - [Implementation Considerations](#)
 - [Binary Distributions](#)
 - [Create the Accounting Structure](#)
 - [Downloading Apache](#)
 - [Compiling Apache](#)
 - [Installing Apache](#)
 - [Configuring Apache](#)
 - [Running Apache](#)
 - [Controlling Apache](#)
-

Requirements

Apache 1.3 requires MPE/iX 6.0 or greater. It will NOT run on earlier releases of MPE/iX. The following MPE/iX patches (or their superseding descendants) are relevant to Apache:

- MPE/iX 6.0:
 - ○ MPEKXT3B - fixes an MPE bug that results in transient "permission denied" errors being returned by the server to the browser.
 - MPELX36A - enhances the kill() function so that MPE users with SM capability can send signals to Apache for shutdown, restart, etc.
 - MPELX44C - fixes an MPE bug that prevents DSO modules from being dynamically loaded.
 - MPELX51C - enhances the kill() function so that Apache can use it when the Apache parent UID is different from the Apache children UID (strongly recommended).
 - NSTxxxxx - the latest network transport patch should always be installed when using TCP/IP applications such as Apache.
- MPE/iX 6.5:

- ○ MPELX44D - fixes an MPE bug that prevents DSO modules from being dynamically loaded.
 - MPELX51D - enhances the kill() function so that Apache can use it when the Apache parent UID is different from the Apache children UID (strongly recommended).
 - NSTxxxxx - the latest network transport patch should always be installed when using TCP/IP applications such as Apache.
-

Implementation Considerations

While MPE has a very good POSIX implementation that enables fairly simple porting of Unix applications such as Apache, there are some Unix concepts which just don't exist or aren't fully implemented in MPE, and so this may force some functionality changes in the package being ported.

Significant MPE vs. Unix OS differences

- MPE lacks the concept of a Unix UID=0 root user with special privileges. Where Unix functions require a user to be executing as root, MPE requires the user to be executing in priv mode, so the program file must be linked with PM (Priv Mode) capability, and the Unix function calls must be bracketed by GETPRIVMODE() and GETUSERMODE() calls. The following Unix functions used by Apache are affected:
 - ○ bind() for ports less than 1024
 - setgid()
 - setuid()
- MPE's support for UIDs and GIDs is more limited than Unix. Every MPE account maps to a unique GID. Each MPE account can contain multiple MPE users, and every MPE user maps to a unique UID (UID 0 is not supported). The current UID for a process must correspond to an MPE user within the MPE account that corresponds to the current GID of the process.
- MPE child processes cannot survive the death of their parent. When the parent terminates, any remaining children will be killed.
- MPE doesn't initialize the envp parameter when invoking the main() of a new process. Use the global variable environ instead of envp.
- MPE link() exists, but always returns EIMPL. Use rename() or symlinks instead of hard links.
- MPE doesn't allow the @ character in filenames.
- MPE lacks support for TCP_NODELAY, but that's the default anyway.
- MPE lacks support for SO_KEEPALIVE.
- MPE lacks support for process groups.
- MPE inetd only passes stdin (and NOT stdout) to the invoked service. But you can write to stdin just fine.

Major Apache functionality issues

- Beginning with HP-supported Apache 1.3.9 and HP WebWise MPE/iX Secure Web Server A.01.00 (based on Apache 1.3.9), the User and Group directives in httpd.conf are now unconditionally executed as corresponding setuid()/setgid() calls. Previously this was only done if HTTPD was being run as MANAGER.SYS. This functionality change was submitted back to the 1.3.13-dev source tree at www.apache.org. The Apache for Unix behavior is to only honor User and Group if running as root.
- Beginning with HP-supported Apache 1.3.9 and HP WebWise MPE/iX Secure Web Server A.01.00 (based on Apache 1.3.9), the SVIPC shared memory macros SHM_R and SHM_W have been modified from their traditional owner-only-read and owner-only-write values to be owner-and-group-read and owner-and-group-write on MPE/iX in order to allow increased parent/child flexibility in spite of MPE's limited POSIX UID/GID support. This functionality change was submitted back to the 1.3.13-dev source tree at www.apache.org. The Apache for Unix behavior uses the traditional owner-only values of SHM_R and SHM_W.

Minor Apache functionality issues

- Apache for Unix must be run as root to bind to TCP ports 1-1023. Apache for MPE must call GETPRIVMODE() to bind to TCP ports 1-1023; PM is not used for ports greater than 1023. The standard web server HTTP port is 80.
 - Apache for Unix in standalone mode will detach itself and run in the background as a system-type process. Apache for MPE in standalone mode cannot detach itself and run in the background because MPE POSIX doesn't allow this (the detached child would be killed when the parent terminated). Therefore you must use an MPE batch job to run Apache in standalone mode.
 - Apache for Unix uses process groups to manage child processes. Apache for MPE cannot use process groups because MPE POSIX doesn't support this. The implications of this are unknown.
 - Apache for Unix uses the setsockopt() option TCP_NODELAY. Apache for MPE does not, because MPE doesn't support it. But TCP_NODELAY is the default MPE behavior anyway.
 - Apache for Unix uses the setsockopt() option SO_KEEPALIVE. Apache for MPE does not, because MPE doesn't support it.
 - Apache for Unix under inetd reads from the socket via stdin and writes via stdout. Apache for MPE under inetd reads *AND* writes the socket via stdin. I consider MPE 5.5 inetd to be broken and poorly documented, so I submitted SR 5003355016 to address this. If HP ever alters the MPE inetd to pass the socket the way HPUX inetd does (not likely in the grand scheme of things), the existing Apache for MPE code will break.
 - Apache for Unix will use the @ character in proxy cache filenames, but since @ is illegal in MPE filenames, Apache for MPE uses the % character instead.
-

Binary Distributions

HP ships a fully supported Apache binary distribution with the Fundamental Operating System (FOS) in MPE/iX 6.5 and later. This distribution can be found in the APACHE account.

HP supplies fully supported Apache binary distributions for MPE/iX 6.0 or later available for downloading from <http://jazz.external.hp.com/src/apache/>.

Mark Bixby supplies Apache binary distributions for MPE/iX available for downloading from <http://www.bixby.org/mark/apacheix.html>. Binaries from bixby.org are NOT supported by HP. HP only supports binaries distributed by HP.

All of the binary distributions mentioned above may possibly include functionality that hasn't yet been submitted back to the Apache Software Foundation (though submitting back is the intended goal). Please read the documentation that comes with these binaries in order to determine functionality differences (if any) compared to the latest sources available from the ASF.

If you will be using one of these binary distributions, please stop reading this document and start reading the specific distribution documentation for installation details.

Create the Accounting Structure

Apache can be installed under the account of your choice. For the purposes of this document, the APACHE account will be used:

1. :HELLO MANAGER.SYS
2. :NEWACCT APACHE ,MGR
3. :ALTACCT APACHE ; PASS=xxxxxxxx ; CAP=AM , AL , ND , SF , BA , IA , PM , PH
4. :ALTGROUP PUB .APACHE ; CAP=BA , IA , PM , PH ; ACCESS=(R , L , X : AC ; W , A , S : AL)

5. :ALTUSER MGR.APACHE;CAP=AM,AL,ND,SF,BA,IA,PM,PH;HOME=PUB
6. :NEWUSER SERVER.APACHE
7. :ALTUSER SERVER.APACHE;CAP=ND,SF,BA,IA,PH;HOME=PUB

Downloading Apache

Use your web browser to download the Apache source tarball from <http://www.apache.org/dist/httpd/>. Then ftp upload the tarball to your e3000 as show below:

```
C:\Temp>ftp 3000.host.name
Connected to 3000.host.name.
220 HP ARPA FTP Server [A0009H09] (C) Hewlett-Packard Co. 1990
User (3000.host.name:(none)): MGR.APACHE
331 Password required for MGR.APACHE. Syntax: acctpass
Password:xxxxxxxx
230 User logged on
ftp> quote type L 8
200 Type set to L (byte size 8).
ftp> put apache_v.u.ff.tar.Z /tmp/apache.tar.Z
200 PORT command ok.
150 File: /tmp/apache.tar.Z opened; data connection will be opened
226 Transfer complete.
ftp: 2685572 bytes sent in 2.75Seconds 976.57Kbytes/sec.
ftp> quit
221 Server is closing command connection
```

Unpack the tarball:

1. :HELLO MGR.APACHE
2. :XEQ SH.HPBIN.SYS -L
3. \$ mkdir src
4. \$ chmod 700 src
5. \$ cd src
6. \$ tar xvfopz /tmp/apache.tar.Z

Compiling Apache

It is **STRONGLY** recommended to use gcc instead of the HP C/iX compiler. You can obtain gcc from <http://jazz.external.hp.com/src/gnu/gnuframe.html>.

1. \$ cd apache_v.uu.ff
2. \$./configure --prefix=/APACHE/PUB --enable-module=xxx --enable-module=yyy
...etc...
3. \$ make

Installing Apache

1. \$ make install
2. \$ cd /APACHE/PUB
3. \$ mv bin/httpd HTTPD
4. \$ ln -s HTTPD bin/httpd

```
5. $ callci "xeq linkedit.pub.sys 'altprog HTTPD;cap=ia,ba,ph,pm'"
```

Configuring Apache

Edit /APACHE/PUB/conf/httpd.conf and customize as needed for your environment. Be sure to make the following mandatory changes:

- User SERVER.APACHE
- Group APACHE

Running Apache

Simply create and :STREAM the following standalone server job in order to start Apache:

```
!JOB JHTTPD,MGR.APACHE;OUTCLASS=,2
!XEQ SH.HPBIN.SYS "-c 'umask 007; ./HTTPD -f /APACHE/PUB/conf/httpd.conf'"
!eoj
```

Controlling Apache

Log on as MGR.APACHE (or MANAGER.SYS or any other SM user if you've installed MPELX36A on 6.0) in order to shutdown or restart Apache via the use of signals.

To shut down Apache from the POSIX shell:

```
$ kill `cat /APACHE/PUB/logs/httpd.pid`
```

To shut down Apache from the CI:

```
:XEQ SH.HPBIN.SYS '-c "kill `cat /APACHE/PUB/logs/httpd.pid`"'
```

Apache HTTP Server





Apache HTTP Server Version 1.3

Using Apache With Novell NetWare

This document explains how to install, configure and run Apache 1.3 under Novell NetWare 5.x and above. If you find any bugs, or wish to contribute in other ways, please use our [bug reporting page](#).

The bug reporting page and new-httpd mailing list are *not* provided to answer questions about configuration or running Apache. Before you submit a bug report or request, first consult this document, the [Frequently Asked Questions](#) page and the other relevant documentation topics. If you still have a question or problem, post it to the [novell.devsup.webserver](#) newsgroup, where many Apache users are more than willing to answer new and obscure questions about using Apache on NetWare.

Most of this document assumes that you are installing Apache from a binary distribution. If you want to compile Apache yourself (possibly to help with development, or to track down bugs), see the section on [Compiling Apache for NetWare](#) below.

-
- [Requirements](#)
 - [Downloading Apache for NetWare](#)
 - [Installing Apache for NetWare](#)
 - [Running Apache for NetWare](#)
 - [Configuring Apache for NetWare](#)
 - [Compiling Apache for NetWare](#)
-

Requirements

Apache 1.3 is designed to run on NetWare 5.x and above and is installed by default on all NetWare 6 servers.

If running on NetWare 5.0 you must install Service Pack 5 or above.

If running on NetWare 5.1 you must install Service Pack 1 or above.

NetWare service packs are available [here](#).

Downloading Apache for NetWare

Information on the latest version of Apache can be found on the Apache web server at <http://www.apache.org/>. This will list the current release, any more recent alpha or beta-test releases, together with details of mirror web and anonymous ftp sites.

Installing Apache for NetWare

There is no Apache install program for NetWare currently. You will need to compile apache and copy the files over to the server manually. An install program will be posted at a later date. If you are running NetWare 6, Apache for NetWare has been installed by default.

Follow these steps to install Apache on NetWare from the binary download (assuming you will install to sys:/apache):

- Unzip the binary download file to the root of the SYS: volume (may be installed to any volume)

- Edit the httpd.conf file setting ServerRoot and ServerName to reflect your correct server settings
- Add SYS:/APACHE to the search path. EXAMPLE: SEARCH ADD SYS:\APACHE

Follow these steps to install Apache on NetWare manually from your own build source (assuming you will install to sys:/apache):

- Create a directory called Apache on a NetWare volume
- Copy Apache.nlm, Apachec.nlm, htdigest.nlm, httpasswd.nlm, xmlparse.nlm, and xmltok.nlm to sys:/apache
- Create a directory under SYS:/APACHE called CONF
- Copy all the *.CONF-DIST-NW files to the SYS:/APACHE/CONF directory and rename them all as *.CONF files
- Copy the MIME.TYPES and magic files to SYS:/APACHE/CONF directory
- Copy all files and subdirectories in \apache-1.3\icons to SYS:/APACHE/ICONS
- Create the directory SYS:/APACHE/LOGS on the server
- Create the directory SYS:/APACHE/CGI-BIN on the server
- Create the directory SYS:/APACHE/MODULES and copy all nlm modules built into the modules directory
- Edit the HTTPD.CONF file setting ServerRoot and ServerName to reflect your correct server settings
- Add SYS:/APACHE to the search path. EXAMPLE: SEARCH ADD SYS:\APACHE

Apache may be installed to other volumes besides the default sys volume.

Running Apache for NetWare

To start Apache just type **apache** at the console. This will load apache in the OS address space. If you prefer to load Apache in a protected address space you may specify the address space with the load statement as follows:

```
load address space = apache apache
```

This will load Apache into an address space called apache. Running multiple instances of Apache concurrently on NetWare is possible by loading each instance into its own protected address space.

After starting Apache it will be listening to port 80 (unless you changed the Port, Listen or BindAddress directives in the configuration files). To connect to the server and access the default page, launch a browser and enter the server's name or address. This should respond with a welcome page, and a link to the Apache manual. If nothing happens or you get an error, look in the error_log file in the logs directory.

Once your basic installation is working, you should configure it properly by editing the files in the conf directory.

To unload Apache running in the OS address space just type the following at the console:

```
unload apache
```

If apache is running in a protected address space specify the address space in the unload statement:

```
unload address space = apache apache
```

When working with Apache it is important to know how it will find the configuration files. You can specify a configuration file on the command line in two ways:

- -f specifies a path to a particular configuration file

```
apache -f "vol:/my server/conf/my.conf"
```

```
apache -f test/test.conf
```

In these cases, the proper ServerRoot should be set in the configuration file.

If you don't specify a configuration file name with -f, Apache will use the file name compiled into the server, usually "conf/httpd.conf". Invoking Apache with the -V switch will display this value labeled as SERVER_CONFIG_FILE. Apache will

then determine its ServerRoot by trying the following, in this order:

- A ServerRoot directive via a -C switch.
- The -d switch on the command line.
- Current working directory
- The server root compiled into the server.

The server root compiled into the server is usually "sys:/apache". Invoking apache with the -V switch will display this value labeled as HTTPD_ROOT.

Configuring Apache for NetWare

Apache is configured by files in the conf directory. These are the same as files used to configure the Unix version, but there are a few different directives for Apache on NetWare. See the [Apache documentation](#) for all the available directives.

The main differences in Apache for NetWare are:

- Because Apache for NetWare is multithreaded, it does not use a separate process for each request, as Apache does with Unix. Instead there are only threads running: a parent thread, and a child which handles the requests. Within the child each request is handled by a separate thread.

So the "process"-management directives are different:

[MaxRequestsPerChild](#) - Like the Unix directive, this controls how many requests a process will serve before exiting. However, unlike Unix, a process serves all the requests at once, not just one, so if this is set, it is recommended that a very high number is used. The recommended default, `MaxRequestsPerChild 0`, does not cause the process to ever exit.

[ThreadsPerChild](#) - This directive is new, and tells the server how many threads it should use. This is the maximum number of connections the server can handle at once; be sure and set this number high enough for your site if you get a lot of hits. The recommended default is `ThreadsPerChild 50`.

[ThreadStackSize](#) - This directive tells the server what size of stack to use for the individual threads. The recommended default is `ThreadStackSize 65536`.

- The directives that accept filenames as arguments now must use NetWare filenames instead of Unix ones. However, because Apache uses Unix-style names internally, you must use forward slashes, not backslashes. Volumes can be used; if omitted, the drive with the Apache executable will be assumed.
- Apache for NetWare has the ability to load modules at runtime, without recompiling the server. If Apache is compiled normally, it will install a number of optional modules in the `\Apache\modules` directory. To activate these, or other modules, the new [LoadModule](#) directive must be used. For example, to activate the status module, use the following (in addition to the status-activating directives in `access.conf`):

```
LoadModule status_module modules/status
```

Information on [creating loadable modules](#) is also available.

Compiling Apache for NetWare

Compiling Apache requires MetroWerks CodeWarrior 4.04 or higher to be properly installed.

First, unpack the Apache distribution into an appropriate directory. Then go to the `src` subdirectory of the Apache distribution and unzip `ApacheNW.mcp.gz`. You may use a recent version of WinZip to accomplish this or `gzip` for Windows. The main Metrowerks project file for Apache (`ApacheNW.mcp`) is now ready to use. Just double click on it from within explorer and it should automatically launch MetroWerks CodeWarrior.

All major pieces of Apache may be built using the `ApacheNW.mcp` project file. This includes modules such as status, info, and proxy.

Once Apache has been built, it needs to be installed in its server root directory. The default is the `sys:/Apache` directory.

Before running the server you must fill out the conf directory. Copy the `*.conf-dist-nw` from the distribution conf directory and rename `*.conf`. Edit the `ServerRoot` entries to your actual server root (for example `"sys:/apache"`). Copy over the `conf/magic` and `conf/mime.types` files as well.

Apache HTTP Server Version 1.3





Apache HTTP Server

Compiling Apache under UnixWare

To compile a working copy of Apache under UnixWare, there are several other steps you may need to take. These prevent such problems as zombie processes, bind errors, and accept errors, to name a few.

UnixWare 1.x

Make sure that `USE_FCNTL_SERIALIZE_ACCEPT` is defined (if not defined by Apache autoconfiguration). If using the UnixWare `cc` compiler, and you still see `accept()` errors, don't use compiler optimization, or get `gcc`.

UnixWare 2.0.x

SCO patch [tf2163](#) is required in order for Apache to work correctly on UnixWare 2.0.x. See <http://www.sco.com> for UnixWare patch information.

In addition, make sure that `USE_FCNTL_SERIALIZE_ACCEPT` is defined (if not defined by Apache autoconfiguration). To reduce instances of connections in `FIN_WAIT_2` state, you may also want to define `NO_LINGCLOSE` (Apache 1.2 only).

UnixWare 2.1.x

SCO patch [ptf3123](#) is required in order for Apache to work correctly on UnixWare 2.1.x. See <http://www.sco.com> for UnixWare patch information.

NOTE: Unixware 2.1.2 and later already have patch `ptf3123` included

In addition, make sure that `USE_FCNTL_SERIALIZE_ACCEPT` is defined (if not defined by Apache autoconfiguration). To reduce instances of connections in `FIN_WAIT_2` state, you may also want to define `NO_LINGCLOSE` (Apache 1.2 only).

Thanks to Joe Doupnik <JRD@cc.usu.edu> and Rich Vaughn <rvaughn@aad.com> for additional info for UnixWare builds.

Apache HTTP Server





Apache HTTP Server Version 1.3

Running a High-Performance Web Server for BSD

Like other OS's, the listen queue is often the **first limit hit**. The following are comments from "Aaron Gifford <agifford@InfoWest.COM>" on how to fix this on BSDI 1.x, 2.x, and FreeBSD 2.0 (and earlier):

Edit the following two files:

```
/usr/include/sys/socket.h
/usr/src/sys/sys/socket.h
```

In each file, look for the following:

```
/*
 * Maximum queue length specifiable by listen.
 */
#define SOMAXCONN      5
```

Just change the "5" to whatever appears to work. I bumped the two machines I was having problems with up to 32 and haven't noticed the problem since.

After the edit, recompile the kernel and recompile the Apache server then reboot.

FreeBSD 2.1 seems to be perfectly happy, with SOMAXCONN set to 32 already.

Addendum for *very heavily loaded BSD servers*

from Chuck Murcko <chuck@telebase.com>

If you're running a really busy BSD Apache server, the following are useful things to do if the system is acting sluggish:

- Run vmstat to check memory usage, page/swap rates, *etc.*
- Run netstat -m to check mbuf usage
- Run fstat to check file descriptor usage

These utilities give you an idea what you'll need to tune in your kernel, and whether it'll help to buy more RAM. Here are some BSD kernel config parameters (actually BSDI, but pertinent to FreeBSD and other 4.4-lite derivatives) from a system getting heavy usage. The tools mentioned above were used, and the system memory was increased to 48 MB before these tuneups. Other system parameters remained unchanged.

```
maxusers      256
```

Maxusers drives a *lot* of other kernel parameters:

- Maximum # of processes
- Maximum # of processes per user
- System wide open files limit
- Per-process open files limit
- Maximum # of mbuf clusters
- Proc/pgrp hash table size

The actual formulae for these derived parameters are in `/usr/src/sys/conf/param.c`. These calculated parameters can also be overridden (in part) by specifying your own values in the kernel configuration file:

```
# Network options. NMBCLUSTERS defines the number of mbuf clusters and
```

```
# defaults to 256. This machine is a server that handles lots of traffic,
# so we crank that value.
```

```
options          NMBCLUSTERS=4096          # mbuf clusters at 4096

#
# Misc. options
#
options          CHILD_MAX=512            # maximum number of child processes
options          OPEN_MAX=512            # maximum fds (breaks RPC svcs)
```

In many cases, NMBCLUSTERS must be set much larger than would appear necessary at first glance. The reason for this is that if the browser disconnects in mid-transfer, the socket fd associated with that particular connection ends up in the TIME_WAIT state for several minutes, during which time its mbufs are not yet freed. Another reason is that, on server timeouts, some connections end up in FIN_WAIT_2 state forever, because this state doesn't time out on the server, and the browser never sent a final FIN. For more details see the [FIN_WAIT_2](#) page.

Some more info on mbuf clusters (from sys/mbuf.h):

```
/*
 * Mbufs are of a single size, MSIZE (machine/machparam.h), which
 * includes overhead. An mbuf may add a single "mbuf cluster" of size
 * MCLBYTES (also in machine/machparam.h), which has no additional overhead
 * and is used instead of the internal data area; this is done when
 * at least MINCLSIZE of data must be stored.
 */
```

CHILD_MAX and OPEN_MAX are set to allow up to 512 child processes (different than the maximum value for processes per user ID) and file descriptors. These values may change for your particular configuration (a higher OPEN_MAX value if you've got modules or CGI scripts opening lots of connections or files). If you've got a lot of other activity besides httpd on the same machine, you'll have to set NPROC higher still. In this example, the NPROC value derived from maxusers proved sufficient for our load.

To increase the size of the listen() queue, you need to adjust the value of SOMAXCONN. SOMAXCONN is not derived from maxusers, so you'll always need to increase that yourself. We use a value guaranteed to be larger than Apache's default for the listen() of 128, currently. The actual value for SOMAXCONN is set in sys/socket.h. The best way to adjust this parameter is run-time, rather than changing it in this header file and thus hardcoding a value in the kernel and elsewhere. To do this, edit /etc/rc.local and add the following line:

```
/usr/sbin/sysctl -w kern.somaxconn=256
```

We used 256 but you can tune it for your own setup. In many cases, however, even the default value of 128 (for later versions of FreeBSD) is OK.

Caveats

Be aware that your system may not boot with a kernel that is configured to use more resources than you have available system RAM. **ALWAYS** have a known bootable kernel available when tuning your system this way, and use the system tools beforehand to learn if you need to buy more memory before tuning.

RPC services will fail when the value of OPEN_MAX is larger than 256. This is a function of the original implementations of the RPC library, which used a byte value for holding file descriptors. BSDI has partially addressed this limit in its 2.1 release, but a real fix may well await the redesign of RPC itself.

Finally, there's the hard limit of child processes configured in Apache.

For versions of Apache later than 1.0.5 you'll need to change the definition for **HARD_SERVER_LIMIT** in *httpd.h* and recompile if you need to run more than the default 150 instances of httpd.

From conf/httpd.conf-dist:

```
# Limit on total number of servers running, i.e., limit on the number
# of clients who can simultaneously connect --- if this limit is ever
```

```
# reached, clients will be LOCKED OUT, so it should NOT BE SET TOO LOW.  
# It is intended mainly as a brake to keep a runaway server from taking  
# Unix with it as it spirals down...
```

MaxClients 150

Know what you're doing if you bump this value up, and make sure you've done your system monitoring, RAM expansion, and kernel tuning beforehand. Then you're ready to service some serious hits!

Thanks to *Tony Sanders* and *Chris Torek* at BSDI for their helpful suggestions and information.

"M. Teterin" <mi@ALDAN.ziplink.net> writes:

```
It really does help if your kernel and frequently used utilities are fully optimized. Rebuilding the FreeBSD kernel on  
an AMD-133 (486-class CPU) web-server with  
-m486 -fexpensive-optimizations -fomit-frame-pointer -O2  
helped reduce the number of "unable" errors, because the CPU was often maxed out.
```

Accept filtering on FreeBSD

Versions of FreeBSD from August 2000 onwards include a feature called "accept filters" which delay the return from `accept()` until a condition has been met, e.g. an HTTP request has arrived. This postpones the requirement for a child process to handle the new connection which therefore increases the number of connections that a given number of child processes can handle. It also allows a child process to accomplish more immediately after `accept()` returns (because the request is already available to be read) so there is less context switching.

There are two filters in FreeBSD at the time of writing: "dataready" and "httpready". The former just waits for the first packet to arrive from the client; the latter waits for the end of the HTTP headers. Unfortunately the "httpready" filter breaks support for HTTP/0.9 (which doesn't have headers) so Apache doesn't use it, but the "dataready" filter gives the same benefit in the majority of cases so Apache attempts to use that instead.

Accept filters provide the most benefit on servers that are already so busy that they are configured with "KeepAlive Off". [HTTP KeepAlive \(aka persistent connections\)](#) avoids the cost of setting up a new connection for every request, but connections that are being kept alive use up one of the available child processes. Since there is a limited number of child processes this can significantly reduce the capacity of the server. The viewers of a web site will still get a lot of the benefit of persistent connections even with a very small `KeepAliveTimeout` so you should try reducing it before turning it off altogether.

To enable accept filtering, you must either load the appropriate accept filter module, e.g. with the command `kldload accf_data`, or compile a kernel with options `ACCEPT_FILTER_DATA`. Apache will then enable filtering when it is restarted.

Accept filters are compiled in if the symbol `SO_ACCEPTFILTER` is defined on the machine apache is build.

Additionally there is a directive [AcceptFilter](#) to switch the filters on or off. The default is on; except when apache is compiled with `-D AP_ACCEPTFILTER_ON`.

If you are more concerned about performance than compatibility with absurdly obsolete HTTP/0.9 user agents then you can recompile Apache to use the "httpready" filter. This may be particularly helpful if your web site uses really big cookies, for example. If you are using `src/Configure` then add `-DACCEPT_FILTER_NAME=\"httpready\"` to the `EXTRA_CFLAGS` line in the `src/Configuration` file. If you are using APACI (aka `./configure`) then use the command `CFLAGS=-DACCEPT_FILTER_NAME=\"httpready\"` `./configure` (with all the funky backslashed quotes).

More welcome!

If you have tips to contribute, send mail to apache@apache.org

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Performance Tuning Tips for Digital Unix

Below is a set of newsgroup posts made by an engineer from DEC in response to queries about how to modify DEC's Digital Unix OS for more heavily loaded web sites. Copied with permission.

Update

From: Jeffrey Mogul <mogul@pa.dec.com>

Date: Fri, 28 Jun 96 16:07:56 MDT

1. The advice given in the README file regarding the "tcbhashsize" variable is incorrect. The largest value this should be set to is 1024. Setting it any higher will have the perverse result of disabling the hashing mechanism.
2. Patch ID OSF350-146 has been superseded by
 - Patch ID OSF350-195 for V3.2C
 - Patch ID OSF360-350195 for V3.2D

Patch IDs for V3.2E and V3.2F should be available soon. There is no known reason why the Patch ID OSF360-350195 won't work on these releases, but such use is not officially supported by Digital. This patch kit will not be needed for V3.2G when it is released.

From mogul@pa.dec.com (Jeffrey Mogul)
Organization DEC Western Research
Date 30 May 1996 00:50:25 GMT
Newsgroups comp.unix.osf.osf1
Message-ID <4o1rch\$bc8@usenet.pa.dec.com>
Subject Re: Web Site Performance
References 1

In article <skoogDs54BH.9pF@netcom.com> skoog@netcom.com (Jim Skoog) writes:

>Where are the performance bottlenecks for Alpha AXP running the
>Netscape Commerce Server 1.12 with high volume internet traffic?
>We are evaluating network performance for a variety of Alpha AXP
>running DEC UNIX 3.2C, which run DEC's seal firewall and behind
>that Alpha 1000 and 2100 web servers.

Our experience (running such Web servers as altavista.digital.com and www.digital.com) is that there is one important kernel tuning knob to adjust in order to get good performance on V3.2C. You need to patch the kernel global variable "somaxconn" (use dbx -k to do this) from its default value of 8 to something much larger.

How much larger? Well, no larger than 32767 (decimal). And probably no less than about 2048, if you have a really high volume (millions of hits per day), like AltaVista does.

This change allows the system to maintain more than 8 TCP

connections in the SYN_RCVD state for the HTTP server. (You can use "netstat -An |grep SYN_RCVD" to see how many such connections exist at any given instant).

If you don't make this change, you might find that as the load gets high, some connection attempts take a very long time. And if a lot of your clients disconnect from the Internet during the process of TCP connection establishment (this happens a lot with dialup users), these "embryonic" connections might tie up your somaxconn quota of SYN_RCVD-state connections. Until the kernel times out these embryonic connections, no other connections will be accepted, and it will appear as if the server has died.

The default value for somaxconn in Digital UNIX V4.0 will be quite a bit larger than it has been in previous versions (we inherited this default from 4.3BSD).

Digital UNIX V4.0 includes some other performance-related changes that significantly improve its maximum HTTP connection rate. However, we've been using V3.2C systems to front-end for altavista.digital.com with no obvious performance bottlenecks at the millions-of-hits-per-day level.

We have some Webstone performance results available at <http://www.digital.com/info/alphaserver/news/webff.html>

[The document referenced above is no longer at that URL -- Ed.]

I'm not sure if these were done using V4.0 or an earlier version of Digital UNIX, although I suspect they were done using a test version of V4.0.

-Jeff

From mogul@pa.dec.com (Jeffrey Mogul)
Organization DEC Western Research
Date 31 May 1996 21:01:01 GMT
Newsgroups comp.unix.osf.osf1
Message-ID <4onmmd\$mmd@usenet.pa.dec.com>
Subject Digital UNIX V3.2C Internet tuning patch info

Something that probably few people are aware of is that Digital has a patch kit available for Digital UNIX V3.2C that may improve Internet performance, especially for busy web servers.

This patch kit is one way to increase the value of somaxconn, which I discussed in a message here a day or two ago.

I've included in this message the revised README file for this patch kit below. Note that the original README file in the patch kit itself may be an earlier version; I'm told that the version below is the right one.

Sorry, this patch kit is NOT available for other versions of Digital UNIX. Most (but not quite all) of these changes also made it into V4.0, so the description of the various tuning parameters in this README file might be useful to people running V4.0 systems.

This patch kit does not appear to be available (yet?) from http://www.service.digital.com/html/patch_service.html so I guess you'll have to call Digital's Customer Support to get it.

-Jeff

DESCRIPTION: Digital UNIX Network tuning patch

Patch ID: OSF350-146

SUPERSEDED PATCHES: OSF350-151, OSF350-158

This set of files improves the performance of the network subsystem on a system being used as a web server. There are additional tunable parameters included here, to be used cautiously by an informed system administrator.

TUNING

To tune the web server, the number of simultaneous socket connection requests are limited by:

somaxconn	Sets the maximum number of pending requests allowed to wait on a listening socket. The default value in Digital UNIX V3.2 is 8. This patch kit increases the default to 1024, which matches the value in Digital UNIX V4.0.
sominconn	Sets the minimum number of pending connections allowed on a listening socket. When a user process calls listen with a backlog less than sominconn, the backlog will be set to sominconn. sominconn overrides somaxconn. The default value is 1.

The effectiveness of tuning these parameters can be monitored by the sobacklog variables available in the kernel:

sobacklog_hiwat	Tracks the maximum pending requests to any socket. The initial value is 0.
sobacklog_drops	Tracks the number of drops exceeding the socket set backlog limit. The initial value is 0.
somaxconn_drops	Tracks the number of drops exceeding the somaxconn limit. When sominconn is larger than somaxconn, tracks the number of drops exceeding sominconn. The initial value is 0.

TCP timer parameters also affect performance. Tuning the following require some knowledge of the characteristics of the network.

tcp_msl	Sets the tcp maximum segment lifetime. This is the maximum lifetime in half seconds that a packet can be in transit on the network. This value, when doubled, is the length of time a connection remains in the TIME_WAIT state after a incoming close request is processed. The unit is specified in 1/2 seconds, the initial value is 60.
---------	---

tcp_rexmit_interval_min

Sets the minimum TCP retransmit interval. For some WAN networks the default value may be too short, causing unnecessary duplicate packets to be sent. The unit is specified in 1/2 seconds, the initial value is 1.

tcp_keepinit

This is the amount of time a partially established connection will sit on the listen queue before timing out (e.g., if a client sends a SYN but never answers our SYN/ACK). Partially established connections tie up slots on the listen queue. If the queue starts to fill with connections in SYN_RCVD state, tcp_keepinit can be decreased to make those partial connects time out sooner. This should be used with caution, since there might be legitimate clients that are taking a while to respond to SYN/ACK. The unit is specified in 1/2 seconds, the default value is 150 (ie. 75 seconds).

The hashlist size for the TCP inpcb lookup table is regulated by:

tcbhashsize

The number of hash buckets used for the TCP connection table used in the kernel. The initial value is 32. For best results, should be specified as a power of 2. For busy Web servers, set this to 2048 or more.

The hashlist size for the interface alias table is regulated by:

inifaddr_hsize

The number of hash buckets used for the interface alias table used in the kernel. The initial value is 32. For best results, should be specified as a power of 2.

ipport_userreserved

The maximum number of concurrent non-reserved, dynamically allocated ports. Default range is 1025-5000. The maximum value is 65535. This limits the number of times you can simultaneously telnet or ftp out to connect to other systems.

tcpnodelack

Don't delay acknowledging TCP data; this can sometimes improve performance of locally run CAD packages. Default is value is 0, the enabled value is 1.

Digital UNIX version:

Feature	V3.2C	V3.2C patch	V4.0
=====	=====	=====	=====
somaxconn	X	X	X
sominconn	-	X	X
sobacklog_hiwat	-	X	-
sobacklog_drops	-	X	-
somaxconn_drops	-	X	-
tcpnodelack	X	X	X
tcp_keepidle	X	X	X

Performance Tuning Tips for Digital Unix

tcp_keepintvl	X	X	X
tcp_keepcnt	-	X	X
tcp_keepinit	-	X	X
TCP keepalive per-socket	-	-	X
tcp_msl	-	X	-
tcp_rexmit_interval_min	-	X	-
TCP inpcb hashing	-	X	X
tcbhashsize	-	X	X
interface alias hashing	-	X	X
inifaddr_hsize	-	X	X
ipport_userreserved	-	X	-
sysconfig -q inet	-	-	X
sysconfig -q socket	-	-	X

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Running a High-Performance Web Server for HPUX

Date: Wed, 05 Nov 1997 16:59:34 -0800
From: Rick Jones <raj@cup.hp.com>
Reply-To: raj@cup.hp.com
Organization: Network Performance
Subject: HP-UX tuning tips

Here are some tuning tips for HP-UX to add to the tuning page.

For HP-UX 9.X: Upgrade to 10.20
For HP-UX 10.[00|01|10]: Upgrade to 10.20

For HP-UX 10.20:

Install the latest cumulative ARPA Transport Patch. This will allow you to configure the size of the TCP connection lookup hash table. The default is 256 buckets and must be set to a power of two. This is accomplished with adb against the *disc* image of the kernel. The variable name is tcp_hash_size. Notice that it's critically important that you use "W" to write a 32 bit quantity, not "w" to write a 16 bit value when patching the disc image because the tcp_hash_size variable is a 32 bit quantity.

How to pick the value? Examine the output of <ftp://ftp.cup.hp.com/dist/networking/tools/connhist> and see how many total TCP connections exist on the system. You probably want that number divided by the hash table size to be reasonably small, say less than 10. Folks can look at HP's SPECweb96 disclosures for some common settings. These can be found at <http://www.specbench.org/>. If an HP-UX system was performing at 1000 SPECweb96 connections per second, the TIME_WAIT time of 60 seconds would mean 60,000 TCP "connections" being tracked.

Folks can check their listen queue depths with <ftp://ftp.cup.hp.com/dist/networking/misc/listenq>.

If folks are running Apache on a PA-8000 based system, they should consider "chatr'ing" the Apache executable to have a large page size. This would be "chatr +pi L <BINARY>." The GID of the running executable must have MLOCK privileges. Setprivgrp(1m) should be consulted for assigning MLOCK. The change can be validated by running Glance and examining the memory regions of the server(s) to make sure that they show a non-trivial fraction of the text segment being locked.

If folks are running Apache on MP systems, they might consider writing a small program that uses mpctl() to bind processes to processors. A simple pid % numcpu algorithm is probably sufficient. This might even go into the source code.

If folks are concerned about the number of FIN_WAIT_2 connections, they can use net tune to shrink the value of tcp_keepstart. However, they should be careful there - certainly do not make it less than oh two to four minutes. If tcp_hash_size has been set well, it is probably OK to let the FIN_WAIT_2's take longer to timeout (perhaps even the default two hours) - they will not on average have a big impact on performance.

There are other things that could go into the code base, but that might be left for another email. Feel free to drop me a message if you or others are interested.

sincerely,

rick jones
<http://www.cup.hp.com/netperf/NetperfPage.html>

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Hints on Running a High-Performance Web Server

Running Apache on a heavily loaded web server, one often encounters problems related to the machine and OS configuration. "Heavy" is relative, of course - but if you are seeing more than a couple hits per second on a sustained basis you should consult the pointers on this page. In general the suggestions involve how to tune your kernel for the heavier TCP load, hardware/software conflicts that arise, *etc.*

- [A/UX \(Apple's UNIX\)](#)
- [BSD-based \(BSDI, FreeBSD, etc\)](#)
- [Digital UNIX](#)
- [HPUX](#)
- [Linux](#)
- [Solaris](#)
- [SunOS 4.x](#)
- [SVR4](#)

A/UX (Apple's UNIX)

If you are running Apache on A/UX, a page that gives some helpful performance hints (concerning the *listen()* queue and using virtual hosts) [can be found here](#)

BSD-based (BSDI, FreeBSD, etc)

[Quick](#) and [detailed](#) performance tuning hints for BSD-derived systems. [Accept filtering](#) on FreeBSD.

Digital UNIX

- [DIGITAL UNIX Tuning Parameters for Web Servers](#)
- We have some [newsgroup postings](#) on how to tune Digital UNIX 3.2 and 4.0.

Linux

There are no known problems with heavily loaded systems running Linux kernels 2.0.32 or later. Earlier kernels have some problems, and an upgrade to the latest 2.0.x is a good idea to eliminate various security and denial of service attacks.



Apache HTTP Server Version 1.3

Server and Supporting Programs

This page documents all the executable programs included with the Apache HTTP Server.

[httpd](#)

Apache hypertext transfer protocol server

[apachectl](#)

Apache HTTP server control interface

[ab](#)

Apache HTTP server benchmarking tool

[apxs](#)

APache eXtenSion tool

[dbmmanage](#)

Create and update user authentication files in DBM format for basic authentication

[htdigest](#)

Create and update user authentication files for digest authentication

[htpasswd](#)

Create and update user authentication files for basic authentication

[logresolve](#)

Resolve hostnames for IP-addresses in Apache logfiles

[rotatelogs](#)

Rotate Apache logs without having to kill the server

[suexec](#)

Switch User For Exec

[Other Programs](#)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: httpd

NAME

`httpd` - Apache hypertext transfer protocol server

SYNOPSIS

```
httpd [ -X ] [ -R libexecdir ] [ -d serverroot ] [ -f config ] [ -C directive ] [ -c directive ] [ -D parameter ]
```

```
httpd [ -h ] [ -l ] [ -L ] [ -v ] [ -V ] [ -S ] [ -t ] [ -T ]
```

DESCRIPTION

`httpd` is the Apache HyperText Transfer Protocol (HTTP) server program. It is designed to be run as a standalone daemon process. When used like this it will create a pool of child processes to handle requests. To stop it, send a TERM signal to the initial (parent) process. The PID of this process is written to a file as given in the configuration file. Alternatively `httpd` may be invoked by the Internet daemon `inetd(8)` each time a connection to the HTTP service is made.

This manual page only lists the command line arguments. For details of the directives necessary to configure `httpd` see the Apache manual, which is part of the Apache distribution or can be found at <http://www.apache.org/>. Paths in this manual may not reflect those compiled into `httpd`.

OPTIONS

`-R libexecdir`

This option is only available if Apache was built with the `SHARED_CORE` rule enabled which forces the Apache core code to be placed into a dynamic shared object (DSO) file. This file is searched in a hardcoded path under `ServerRoot` per default. Use this option if you want to override it.

`-d serverroot`

Set the initial value for the `ServerRoot` directive to `serverroot`. This can be overridden by the `ServerRoot` command in the configuration file. The default is `/usr/local/apache`.

`-f config`

Execute the commands in the file `config` on startup. If `config` does not begin with a `/`, then it is taken to be a path relative to the `ServerRoot`. The default is `conf/httpd.conf`.

`-C directive`

Process the configuration *directive* before reading config files.

-c *directive*

Process the configuration *directive* after reading config files.

-D *parameter*

Sets a configuration *parameter* which can be used with `<IfDefine>...</IfDefine>` sections in the configuration files to conditionally skip or process commands.

-h

Output a short summary of available command line options.

-l

Output a list of modules compiled into the server.

-L

Output a list of directives together with expected arguments and places where the directive is valid.

-S

Show the settings as parsed from the config file (currently only shows the virtualhost settings).

-t

Run syntax tests for configuration files only. The program immediately exits after these syntax parsing with either a return code of 0 (Syntax OK) or return code not equal to 0 (Syntax Error).

-T

Same as option **-t** but does not check the configured document roots.

-X

Run in single-process mode, for internal debugging purposes only; the daemon does not detach from the terminal or fork any children. Do NOT use this mode to provide ordinary web service.

-v

Print the version of **httpd** , and then exit.

-V

Print the version and build parameters of **httpd** , and then exit.

FILES

/usr/local/apache/conf/httpd.conf
/usr/local/apache/conf/srm.conf
/usr/local/apache/conf/access.conf
/usr/local/apache/conf/mime.types
/usr/local/apache/conf/magic
/usr/local/apache/logs/error_log
/usr/local/apache/logs/access_log
/usr/local/apache/logs/httpd.pid

SEE ALSO

inetd(8).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: apxs

NAME

apxs - APache eXtenSion tool

SYNOPSIS

```
apxs -g [ -S variable=value ] -n name
```

```
apxs -q [ -S variable=value ] query ...
```

```
apxs -c [ -S variable=value ] [ -o dsofile ] [ -I incdir ] [
-D variable[=value] ] [ -L libdir ] [ -l libname ] [
-Wc,compiler-flags ] [ -Wl,linker-flags ] files ...
```

```
apxs -i [ -S variable=value ] [ -n name ] [ -a ] [ -A ] dso-
file ...
```

```
apxs -e [ -S variable=value ] [ -n name ] [ -a ] [ -A ] dso-
file ...
```

DESCRIPTION

apxs is a tool for building and installing extension modules for the Apache HyperText Transfer Protocol (HTTP) server. This is achieved by building a Dynamic Shared Object (DSO) from one or more source or object *files* which then can be loaded into the Apache server under runtime via the **LoadModule** directive from **mod_so**.

So to use this extension mechanism, your platform has to support the DSO feature and your Apache **httpd** binary has to be built with the **mod_so** module. The **apxs** tool automatically complains if this is not the case. You can check this yourself by manually running the command

```
$ httpd -l
```

The module **mod_so** should be part of the displayed list. If these requirements are fulfilled, you can easily extend your Apache server's functionality by installing your own modules with the DSO mechanism by the help of this **apxs** tool:

```
$ apxs -i -a -c mod_foo.c
gcc -fpic -DSHARED_MODULE -I/path/to/apache/include -c mod_foo.c
ld -Bshareable -o mod_foo.so mod_foo.o
cp mod_foo.so /path/to/apache/libexec/mod_foo.so
chmod 755 /path/to/apache/libexec/mod_foo.so
[activating module `foo' in /path/to/apache/etc/httpd.conf]
$ apachectl restart
/path/to/apache/sbin/apachectl restart: httpd not running, trying to start
[Tue Mar 31 11:27:55 1998] [debug] mod_so.c(303): loaded module foo_module
/path/to/apache/sbin/apachectl restart: httpd started
```

\$ _

The arguments *files* can be any C source file (.c), a object file (.o) or even a library archive (.a). The **apxs** tool automatically recognizes these extensions and automatically uses the C source files for compilation while it just uses the object and archive files for the linking phase. But when using such pre-compiled objects, make sure they are compiled for Position Independent Code (PIC) to be able to use them for a DSO. For instance with GCC you always just have to use **-fpic**. For other C compilers please consult its manual page or watch for the flags **apxs** uses to compile the object files.

For more details about DSO support in Apache, first read the background information about DSO in `htdocs/manual/dso.html`, then read the documentation of **mod_so**.

OPTIONS

Common options:

-n name This explicitly sets the module name for the **-i** (install) and **-g** (template generation) option. Use this to explicitly specify the module name. For option **-g** this is required, for option **-i** the **apxs** tool tries to determine the name from the source or (as a fallback) at least by guessing it from the filename.

Query options:

-q Performs a query for **apxs**'s knowledge about certain settings. The *query* parameters can be one or more of the following variable names:

CC	TARGET
CFLAGS	SBINDIR
CFLAGS_SHLIB	INCLUDEDIR
LD_SHLIB	LIBEXECDIR
LD_FLAGS_SHLIB	SYSCONFDIR
LIBS_SHLIB	PREFIX

Use this for manually determining settings. For instance use

```
INC=-I`apxs -q INCLUDEDIR`
```

inside your own Makefiles if you need manual access to Apache's C header files.

Configuration options:

-s variable=value This option changes the **apxs** settings described above.

Template Generation options:

-g This generates a subdirectory *name* (see option **-n**) and there two files: A sample module source file named **mod_name.c** which can be used as a template for creating your own modules or as a quick start for playing with the **apxs** mechanism. And a corresponding **Makefile** for even easier building and installing of this module.

DSO compilation options:

- c This indicates the compilation operation. It first compiles the C source files (.c) of *files* into corresponding object files (.o) and then builds a DSO in *dsofile* by linking these object files plus the remaining object files (.o and .a) of *files*. If no **-o** option is specified the output file is guessed from the first filename in *files* and thus usually defaults to **mod_name.so**.
- o *dsofile* Explicitly specifies the filename of the created DSO file. If not specified and the name cannot be guessed from the *files* list, the fallback name **mod_unknown.so** is used.
- D *variable[=value]* This option is directly passed through to the compilation command(s). Use this to add your own defines to the build process.
- I *incdir* This option is directly passed through to the compilation command(s). Use this to add your own include directories to search to the build process.
- L *libdir* This option is directly passed through to the linker command. Use this to add your own library directories to search to the build process.
- l *libname* This option is directly passed through to the linker command. Use this to add your own libraries to search to the build process.
- Wc,*compiler-flags* This option passes *compiler-flags* as additional flags to the compiler command. Use this to add local compiler-specific options.
- Wl,*linker-flags* This option passes *linker-flags* as additional flags to the linker command. Use this to add local linker-specific options.

DSO installation and configuration options:

- i This indicates the installation operation and installs one or more DSOs into the server's *libexec* directory.
- a This activates the module by automatically adding a corresponding **LoadModule** line to Apache's **httpd.conf** configuration file, or by enabling it if it already exists.
- A Same as option **-a** but the created **LoadModule** directive is prefixed with a hash sign (#), i.e. the module is just prepared for later activation but initially disabled.
- e This indicates the editing operation, which can

be used with the **-a** and **-A** options similarly to the **-i** operation to edit Apache's **httpd.conf** configuration file without attempting to install the module.

EXAMPLES

Assume you have an Apache module named `mod_foo.c` available which should extend Apache's server functionality. To accomplish this you first have to compile the C source into a DSO suitable for loading into the Apache server under runtime via the following command:

```
$ apxs -c mod_foo.c
gcc -fpic -DSHARED_MODULE -I/path/to/apache/include -c mod_foo.c
ld -Bshareable -o mod_foo.so mod_foo.o
$ _
```

Then you have to update the Apache configuration by making sure a **LoadModule** directive is present to load this DSO. To simplify this step **apxs** provides an automatic way to install the DSO in the "libexec" directory and updating the **httpd.conf** file accordingly. This can be achieved by running:

```
$ apxs -i -a mod_foo.c
cp mod_foo.so /path/to/apache/libexec/mod_foo.so
chmod 755 /path/to/apache/libexec/mod_foo.so
[activating module `foo' in /path/to/apache/etc/httpd.conf]
$ _
```

This way a line named

```
LoadModule foo_module libexec/mod_foo.so
```

is added to the configuration file if still not present. If you want to have this operation to be disabled, use the **-A** option, i.e.

```
$ apxs -i -A mod_foo.c
```

For a quick test of the **apxs** mechanism you can create a sample Apache module template plus a corresponding **Makefile** via:

```
$ apxs -g -n foo
Creating [DIR] foo
Creating [FILE] foo/Makefile
Creating [FILE] foo/mod_foo.c
$ _
```

Then you can immediately compile this sample module into a DSO and load it into the Apache server:

```
$ cd foo
$ make all reload
apxs -c mod_foo.c
gcc -fpic -DSHARED_MODULE -I/path/to/apache/include -c mod_foo.c
ld -Bshareable -o mod_foo.so mod_foo.o
apxs -i -a -n "foo" mod_foo.so
cp mod_foo.so /path/to/apache/libexec/mod_foo.so
chmod 755 /path/to/apache/libexec/mod_foo.so
[activating module `foo' in /path/to/apache/etc/httpd.conf]
```

```
apachectl restart
/path/to/apache/sbin/apachectl restart: httpd not running, trying to start
[Tue Mar 31 11:27:55 1998] [debug] mod_so.c(303): loaded module foo_module
/path/to/apache/sbin/apachectl restart: httpd started
$ _
```

You can even use **apxs** to compile complex modules outside the Apache source tree, like PHP3, because **apxs** automatically recognized C source files and object files.

```
$ cd php3
$ ./configure --with-shared-apache=../apache-1.3
$ apxs -c -o libphp3.so mod_php3.c libmodphp3-so.a
gcc -fpic -DSHARED_MODULE -I/tmp/apache/include -c mod_php3.c
ld -Bshareable -o libphp3.so mod_php3.o libmodphp3-so.a
$ _
```

Only C source files are compiled while remaining object files are used for the linking phase.

SEE ALSO

apachectl(1), httpd(8).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: ab

NAME

ab - Apache HTTP server benchmarking tool

SYNOPSIS

```
ab [ -k ] [ -i ] [ -n requests ] [ -t timelimit ] [ -c con-
currency ] [ -p POST file ] [ -A Authenticate
username:password ] [ -P Proxy Authenticate
username:password ] [ -H Custom header ] [ -C Cookie
name=value ] [ -T content-type ] [ -v verbosity ] [ -w
output HTML ] [ -x <table> attributes ] [ -y <tr> attri-
butes ] [ -z <td> attributes ]
[http://]hostname[:port]/path
```

```
ab [ -v ] [ -h ]
```

DESCRIPTION

ab is a tool for benchmarking the performance of your Apache HyperText Transfer Protocol (HTTP) server. It does this by giving you an indication of how many requests per second your Apache installation can serve.

OPTIONS

- k** Enable the HTTP KeepAlive feature; that is, perform multiple requests within one HTTP session. Default is no KeepAlive.
- i** Use an HTTP 'HEAD' instead of the GET method. Cannot be mixed with POST.
- n requests** The number of requests to perform for the benchmarking session. The default is to perform just one single request, which will not give representative benchmarking results.
- t timelimit** The number of seconds to spend benchmarking. Using this option automatically set the number of requests for the benchmarking session to 50000. Use this to benchmark the server for a fixed period of time. By default, there is no timelimit.
- c concurrency** The number of simultaneous requests to perform. The default is to perform one HTTP request at a time, that is, no concurrency.
- p POST file** A file containing data that the program will

send to the Apache server in any HTTP POST requests.

-A *Authorization username:password*

Supply Basic Authentication credentials to the server. The username and password are separated by a single ':', and sent as uuencoded data. The string is sent regardless of whether the server needs it; that is, has sent a 401 Authentication needed.

-p *Proxy-Authorization username:password*

Supply Basic Authentication credentials to a proxy en-route. The username and password are separated by a single ':', and sent as uuencoded data. The string is sent regardless of whether the proxy needs it; that is, has sent a 407 Proxy authentication needed.

-C *Cookie name=value*

Add a 'Cookie:' line to the request. The argument is typically a 'name=value' pair. This option may be repeated.

-p *Header string*

Append extra headers to the request. The argument is typically in the form of a valid header line, usually a colon separated field value pair, for example, 'Accept-Encoding: zip/zop;8bit'.

-T *content-type*

The content-type header to use for POST data.

-v

Sets the verbosity level. Level 4 and above prints information on headers, level 3 and above prints response codes (for example, 404, 200), and level 2 and above prints warnings and informational messages.

-w

Print out results in HTML tables. The default table is two columns wide, with a white background.

-x *attributes*

The string to use as attributes for <table>. Attributes are inserted <table **here** >

-y *attributes*

The string to use as attributes for <tr>.

-z *attributes*

The string to use as attributes for <td>.

-V

Display the version number and exit.

-h

Display usage information.

BUGS

There are various statically declared buffers of fixed length. Combined with inefficient parsing of the command line arguments, the response headers from the server, and

other external inputs, these buffers might overflow.

Ab does not implement HTTP/1.x fully; instead, it only accepts some 'expected' forms of responses.

The rather heavy use of **strstr(3)** by the program may skew performance results, since it uses significant CPU resources. Make sure that performance limits are not hit by **ab** before your server's limit is reached.

SEE ALSO
httpd(8)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: apachectl

NAME

apachectl - Apache HTTP server control interface

SYNOPSIS

apachectl *command* [...]

DESCRIPTION

apachectl is a front end to the Apache HyperText Transfer Protocol (HTTP) server. It is designed to help the administrator control the functioning of the Apache **httpd** daemon.

NOTE: If your Apache installation uses non-standard paths, you will need to edit the **apachectl** script to set the appropriate paths to your PID file and your **httpd** binary. See the comments in the script for details.

The **apachectl** script returns a 0 exit value on success, and >0 if an error occurs. For more details, view the comments in the script.

Full documentation for Apache is available at <http://www.apache.org/>

OPTIONS

The *command* can be any one or more of the following options:

start Start the Apache daemon. Gives an error if it is already running.

stop Stops the Apache daemon.

restart Restarts the Apache daemon by sending it a SIGHUP. If the daemon is not running, it is started. This command automatically checks the configuration files via **configtest** before initiating the restart to make sure Apache doesn't die.

fullstatus Displays a full status report from **mod_status**. For this to work, you need to have **mod_status** enabled on your server and a text-based browser such as *lynx* available on your system. The URL used to access the status report can be set by editing the **STATUSURL** variable in the script.

status Displays a brief status report. Similar to the **fullstatus** option, except that the list of requests currently being served is omitted.

- graceful** Gracefully restarts the Apache daemon by sending it a SIGUSR1. If the daemon is not running, it is started. This differs from a normal restart in that currently open connections are not aborted. A side effect is that old log files will not be closed immediately. This means that if used in a log rotation script, a substantial delay may be necessary to ensure that the old log files are closed before processing them. This command automatically checks the configuration files via **configtest** before initiating the restart to make sure Apache doesn't die.
- configtest** Run a configuration file syntax test. It parses the configuration files and either reports **Syntax Ok** or detailed information about the particular syntax error.
- help** Displays a short help message.

SEE ALSO
[httpd\(8\)](#)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: dbmmanage

NAME

dbmmanage - Create and update user authentication files in DBM format

SYNOPSIS

```
dbmmanage filename [ command ] [ username [ encpasswd ] ]
```

DESCRIPTION

dbmmanage is used to create and update the DBM format files used to store usernames and password for basic authentication of HTTP users. Resources available from the **httpd** Apache webserver can be restricted to just the users listed in the files created by **dbmmanage**. This program can only be used when the usernames are stored in a DBM file. To use a flat-file database see **htpasswd**.

This manual page only lists the command line arguments. For details of the directives necessary to configure user authentication in **httpd** see the Apache manual, which is part of the Apache distribution or can be found at <http://www.apache.org/>.

OPTIONS

filename

The filename of the DBM format file. Usually without the extension *.db*, *.pag*, or *.dir*.

command

This selects the operation to perform:

- add** Adds an entry for *username* to *filename* using the encrypted password *encpassword*.
- adduser** Asks for a password and then adds an entry for *username* to *filename*.
- check** Asks for a password and then checks if *username* is in *filename* and if its password matches the specified one.
- delete** Deletes the *username* entry from *filename*.
- import** Reads *username:password* entries (one per line) from STDIN and adds them to *filename*. The passwords already has to be crypted.
- update** Same as the "adduser" command, except that it makes sure *username* already exists in *filename*.

view Just displays the complete contents of the DBM file.

username The user for which the update operation is performed.

BUGS

One should be aware that there are a number of different DBM file formats in existence, and with all likelihood, libraries for more than one format may exist on your system. The three primary examples are NDBM, the GNU project's GDBM, and Berkeley DB 2. Unfortunately, all these libraries use different file formats, and you must make sure that the file format used by *filename* is the same format that **dbmmanage** expects to see. **dbmmanage** currently has no way of determining what type of DBM file it is looking at. If used against the wrong format, will simply return nothing, or may create a different DBM file with a different name, or at worst, it may corrupt the DBM file if you were attempting to write to it.

dbmmanage has a list of DBM format preferences, defined by the **@AnyDBM::ISA** array near the beginning of the program. Since we prefer the Berkeley DB 2 file format, the order in which **dbmmanage** will look for system libraries is Berkeley DB 2, then NDBM, and then GDBM. The first library found will be the library **dbmmanage** will attempt to use for all DBM file transactions. This ordering is slightly different than the standard **@AnyDBM::ISA** ordering in perl, as well as the ordering used by the simple `dbmopen()` call in Perl, so if you use any other utilities to manage your DBM files, they must also follow this preference ordering. Similar care must be taken if using programs in other languages, like C, to access these files.

Apache's **mod_auth_db.c** module corresponds to Berkeley DB 2 library, while **mod_auth_dbm.c** corresponds to the NDBM library. Also, one can usually use the **file** program supplied with most Unix systems to see what format a DBM file is in.

SEE ALSO

httpd(8)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: htdigest

NAME

`htdigest` - Create and update user authentication files

SYNOPSIS

```
htdigest [ -c ] passwdfile realm username
```

DESCRIPTION

htdigest is used to create and update the flat-files used to store usernames, realm and password for digest authentication of HTTP users. Resources available from the **httpd** Apache web server can be restricted to just the users listed in the files created by **htdigest**.

This manual page only lists the command line arguments. For details of the directives necessary to configure digest authentication in **httpd** see the Apache manual, which is part of the Apache distribution or can be found at <http://www.apache.org/>.

OPTIONS

`-c` Create the *passwdfile*. If *passwdfile* already exists, it is deleted first.

passwdfile

Name of the file to contain the username, realm and password. If `-c` is specified, this file is created if it does not already exist, or deleted and recreated if it does exist.

realm

The realm name to which the user name belongs.

username

The user name to create or update in *passwdfile*. If *username* does not exist in this file, an entry is added. If it does exist, the password is changed.

SEE ALSO

`httpd(8)`

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: htpasswd

NAME

htpasswd - Create and update user authentication files

SYNOPSIS

```
htpasswd [ -c ] [ -m | -d | -s | -p ] passwdfile username
htpasswd -b [ -c ] [ -m | -d | -s | -p ] passwdfile username
password
htpasswd -n [ -m | -d | -s | -p ] username
htpasswd -nb [ -m | -d | -s | -p ] username password
```

DESCRIPTION

htpasswd is used to create and update the flat-files used to store usernames and password for basic authentication of HTTP users. If **htpasswd** cannot access a file, such as not being able to write to the output file or not being able to read the file in order to update it, it returns an error status and makes no changes.

Resources available from the **httpd** Apache web server can be restricted to just the users listed in the files created by **htpasswd**. This program can only manage usernames and passwords stored in a flat-file. It can encrypt and display password information for use in other types of data stores, though. To use a DBM database see **dbmmanage**.

htpasswd encrypts passwords using either a version of MD5 modified for Apache, or the system's `crypt()` routine. Files managed by **htpasswd** may contain both types of passwords; some user records may have MD5-encrypted passwords while others in the same file may have passwords encrypted with `crypt()`.

This manual page only lists the command line arguments. For details of the directives necessary to configure user authentication in **httpd** see the Apache manual, which is part of the Apache distribution or can be found at [<URL:http://www.apache.org/>](http://www.apache.org/).

OPTIONS

- b Use batch mode; *i.e.*, get the password from the command line rather than prompting for it. **This option should be used with extreme care, since the password is clearly visible on the command line.**
- c Create the *passwdfile*. If *passwdfile* already exists, it is rewritten and truncated. This option cannot be combined with the **-n** option.
- n Display the results on standard output rather than

updating a file. This is useful for generating password records acceptable to Apache for inclusion in non-text data stores. This option changes the syntax of the command line, since the *passwdfile* argument (usually the first one) is omitted. It cannot be combined with the **-c** option.

- m Use Apache's modified MD5 algorithm for passwords. Passwords encrypted with this algorithm are transportable to any platform (Windows, Unix, BeOS, et cetera) running Apache 1.3.9 or later. On Windows and TPF, this flag is the default.
- d Use crypt() encryption for passwords. The default on all platforms but Windows and TPF. Though possibly supported by **htpasswd** on all platforms, it is not supported by the **httpd** server on Windows and TPF.
- s Use SHA encryption for passwords. Faciliates migration from/to Netscape servers using the LDAP Directory Interchange Format (ldif).
- p Use plaintext passwords. Though **htpasswd** will support creation on all platforms, the **httpd** daemon will only accept plain text passwords on Windows and TPF.

passwdfile

Name of the file to contain the user name and password. If **-c** is given, this file is created if it does not already exist, or rewritten and truncated if it does exist.

username

The username to create or update in **passwdfile**. If *username* does not exist in this file, an entry is added. If it does exist, the password is changed.

password

The plaintext password to be encrypted and stored in the file. Only used with the **-b** flag.

EXIT STATUS

htpasswd returns a zero status ("true") if the username and password have been successfully added or updated in the *passwdfile*. **htpasswd** returns 1 if it encounters some problem accessing files, 2 if there was a syntax problem with the command line, 3 if the password was entered interactively and the verification entry didn't match, 4 if its operation was interrupted, 5 if a value is too long (username, filename, password, or final computed record), and 6 if the username contains illegal characters (see the **RESTRICTIONS** section).

EXAMPLES

```
htpasswd /usr/local/etc/apache/.htpasswd-users jsmith
```

Adds or modifies the password for user *jsmith*. The user is prompted for the password. If executed on a Windows system, the password will be encrypted using the modified Apache MD5 algorithm; otherwise, the system's *crypt()* routine will be used. If the file does not exist, **htpasswd** will do nothing except return an error.

```
htpasswd -c /home/does/public_html/.htpasswd jane
```

Creates a new file and stores a record in it for user *jane*. The user is prompted for the password. If the file exists and cannot be read, or cannot be written, it is not altered and **htpasswd** will display a message and return an error status.

```
htpasswd -mb /usr/web/.htpasswd-all jones Pwd4Steve
```

Encrypts the password from the command line (*Pwd4Steve*) using the MD5 algorithm, and stores it in the specified file.

SECURITY CONSIDERATIONS

Web password files such as those managed by **htpasswd** should **not** be within the Web server's URI space -- that is, they should not be fetchable with a browser.

The use of the *-b* option is discouraged, since when it is used the unencrypted password appears on the command line.

RESTRICTIONS

On the Windows and MPE platforms, passwords encrypted with **htpasswd** are limited to no more than 255 characters in length. Longer passwords will be truncated to 255 characters.

The MD5 algorithm used by **htpasswd** is specific to the Apache software; passwords encrypted using it will not be usable with other Web servers.

Usernames are limited to 255 bytes and may not include the character ':'.

SEE ALSO

httpd(8) and the scripts in `support/SHA1` which come with the distribution.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: logresolve

NAME

logresolve - resolve hostnames for IP-addresses in Apache logfiles

SYNOPSIS

```
logresolve [ -s filename ] [ -c ] < access_log >
access_log.new
```

DESCRIPTION

logresolve is a post-processing program to resolve IP-addresses in Apache's access logfiles. To minimize impact on your nameserver, logresolve has its very own internal hash-table cache. This means that each IP number will only be looked up the first time it is found in the log file.

OPTIONS

-s filename Specifies a filename to record statistics.

-c This causes **logresolve** to apply some DNS checks: after finding the hostname from the IP address, it looks up the IP addresses for the hostname and checks that one of these matches the original address.

SEE ALSO

httpd(8)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: rotatelog

NAME

rotatelog - rotate Apache logs without having to kill the server

SYNOPSIS

```
rotatelog logfile rotationtime
```

DESCRIPTION

rotatelog is a simple program for use in conjunction with Apache's piped logfile feature which can be used like this:

```
TransferLog "|rotatelog /path/to/logs/access_log 86400"
```

This creates the files `/path/to/logs/access_log.nnnn` where `nnnn` is the system time at which the log nominally starts (this time will always be a multiple of the rotation time, so you can synchronize cron scripts with it). At the end of each rotation time (here after 24 hours) a new log is started.

OPTIONS

logfile

The path plus basename of the logfile. The suffix `.nnnn` is automatically added.

rotationtime

The rotation time in seconds.

SEE ALSO

`httpd(8)`

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Manual Page: suexec

NAME

suexec - Switch User For Exec

SYNOPSIS

No synopsis for usage, because this program is used internally by Apache only.

DESCRIPTION

suexec is the "wrapper" support program for the suEXEC behavior for Apache. It is run from within Apache automatically to switch the user when an external program has to be run under a different user. For more information about suEXEC see the document 'Apache suEXEC Support' under <http://www.apache.org/docs/suexec.html> .

SEE ALSO

[httpd\(8\)](#)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Other Programs

The following programs are simple support programs included with the Apache HTTP Server which do not have their own manual pages.

log_server_status

This Perl script is designed to be run at a frequent interval by something like cron. It connects to the server and downloads the status information. It reformats the information to a single line and logs it to a file. Adjust the variables at the top of the script to specify the location of the resulting logfile.

split-logfile

This Perl script will take a combined Web server access log file and break its contents into separate files. It assumes that the first field of each line is the virtual host identity (put there by "%v"), and that the logfiles should be named that+".log" in the current directory.

The combined log file is read from stdin. Records read will be appended to any existing log files.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache modules

Below is a list of all of the modules that come as part of the Apache distribution. See also the list of modules [sorted by type](#) and the complete alphabetical list of [all Apache directives](#). For Apache modules that are not part of the Apache distribution, please see <http://modules.apache.org>

[Core](#)

Core Apache features

[mod_access](#)

Access control based on client hostname or IP address

[mod_actions](#) Apache 1.1 and up

Executing CGI scripts based on media type or request method

[mod_alias](#)

Mapping different parts of the host filesystem in the document tree, and URL redirection

[mod_asis](#)

Sending files which contain their own HTTP headers

[mod_auth](#)

User authentication using text files

[mod_auth_anon](#) Apache 1.1 and up

Anonymous user access to authenticated areas

[mod_auth_db](#) Apache 1.1 and up

User authentication using Berkeley DB files

[mod_auth_dbm](#)

User authentication using DBM files

[mod_auth_digest](#) Apache 1.3.8 and up

MD5 authentication

[mod_autoindex](#)

Automatic directory listings

[mod_browser](#) Apache 1.2.* only

Set environment variables based on User-Agent strings. Replaced by `mod_setenvif` in Apache 1.3 and up

[mod_cern_meta](#) Apache 1.1 and up

Support for HTTP header metafiles

[mod_cgi](#)

Invoking CGI scripts

[mod_cookies](#) up to Apache 1.1.1

Support for Netscape-like cookies. Replaced in Apache 1.2 by `mod_usertrack`

[mod_digest](#) Apache 1.1 and up

Apache modules

MD5 authentication (deprecated by mod_auth_digest)

[mod_dir](#)

Basic directory handling

[mod_dld](#) Apache 1.2.* and earlier

Start-time linking with the GNU libdld. Replaced in Apache 1.3 by mod_so

[mod_env](#) Apache 1.1 and up

Passing of environments to CGI scripts

[mod_example](#) Apache 1.2 and up

Demonstrates Apache API

[mod_expires](#) Apache 1.2 and up

Apply Expires: headers to resources

[mod_headers](#) Apache 1.2 and up

Add arbitrary HTTP headers to resources

[mod_imap](#) Apache 1.1 and up

The imagemap file handler

[mod_include](#)

Server-parsed documents

[mod_info](#) Apache 1.1 and up

Server configuration information

[mod_isapi](#) WIN32 only

Windows ISAPI Extension support

[mod_log_agent](#)

Logging of User Agents

[mod_log_common](#) up to Apache 1.1.1

Standard logging in the Common Logfile Format. Replaced by the mod_log_config module in Apache 1.2 and up

[mod_log_config](#)

User-configurable logging replacement for mod_log_common

[mod_log_referer](#)

Logging of document references

[mod_mime](#)

Determining document types using file extensions

[mod_mime_magic](#)

Determining document types using "magic numbers"

[mod_mmap_static](#) Apache 1.3 and up

Experimental file caching, mapping files into memory to improve performance

[mod_negotiation](#)

Content negotiation

[mod_proxy](#) Apache 1.1 and up

Caching proxy abilities

[mod_rewrite](#) Apache 1.2 and up

Powerful URI-to-filename mapping using regular expressions

Apache modules

[mod_setenvif](#) Apache 1.3 and up

Set environment variables based on client information

[mod_so](#) Apache 1.3 and up

Support for loading modules (.so's on Unix, .dll's on Win32) at runtime

[mod_speling](#) Apache 1.3 and up

Automatically correct minor typos in URLs

[mod_status](#) Apache 1.1 and up

Server status display

[mod_unique_id](#) Apache 1.3 and up

Generate unique request identifier for every request

[mod_userdir](#)

User home directories

[mod_usertrack](#) Apache 1.2 and up

User tracking using Cookies (replacement for mod_cookies.c)

[mod_vhost_alias](#) Apache 1.3.7 and up

Support for dynamically configured mass virtual hosting

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache modules

Below is a list of all of the modules that come as part of the Apache distribution. See also the list of modules [sorted alphabetically](#) and the complete alphabetical list of [all Apache directives](#). For modules that are not part of the Apache distribution, please see <http://modules.apache.org>.

Core

[Core](#)

Core Apache features

Environment Creation

[mod_env](#)

Passing of environments to CGI scripts

[mod_setenvif](#) Apache 1.3 and up

Set environment variables based on client information

[mod_unique_id](#) Apache 1.3 and up

Generate unique request identifier for every request

Content Type Decisions

[mod_mime](#)

Determining document types using file extensions

[mod_mime_magic](#)

Determining document types using "magic numbers"

[mod_negotiation](#)

Content negotiation

URL Mapping

[mod_alias](#)

Mapping different parts of the host filesystem in the document tree, and URL redirection

[mod_rewrite](#) Apache 1.2 and up

Powerful URI-to-filename mapping using regular expressions

[mod_userdir](#)

User home directories

Apache modules

[mod_speling](#) Apache 1.3 and up

Automatically correct minor typos in URLs

[mod_vhost_alias](#) Apache 1.3.7 and up

Support for dynamically configured mass virtual hosting

Directory Handling

[mod_dir](#)

Basic directory handling

[mod_autoindex](#)

Automatic directory listings

Access Control

[mod_access](#)

Access control based on client hostname or IP address

[mod_auth](#)

User authentication using text files

[mod_auth_dbm](#)

User authentication using DBM files

[mod_auth_db](#)

User authentication using Berkeley DB files

[mod_auth_anon](#) Apache 1.1 and up

Anonymous user access to authenticated areas

[mod_auth_digest](#) Apache 1.3.8 and up

Experimental MD5 authentication

[mod_digest](#) Apache 1.1 and up

MD5 authentication

HTTP Response

[mod_headers](#) Apache 1.2 and up

Add arbitrary HTTP headers to resources

[mod_cern_meta](#) Apache 1.1 and up

Support for HTTP header metafiles

[mod_expires](#) Apache 1.2 and up

Apply Expires: headers to resources

[mod_asis](#)

Sending files which contain their own HTTP headers

Dynamic Content

[mod_include](#)

Server-parsed documents

[mod_cgi](#)

Invoking CGI scripts

[mod_actions](#) Apache 1.1 and up

Executing CGI scripts based on media type or request method

[mod_isapi](#) WIN32 only

Windows ISAPI Extension support

Internal Content Handlers

[mod_status](#) Apache 1.1 and up

Server status display

[mod_info](#) Apache 1.1 and up

Server configuration information

Logging

[mod_log_config](#)

User-configurable logging replacement for mod_log_common

[mod_log_agent](#)

Logging of User Agents

[mod_log_referer](#)

Logging of document references

[mod_usertrack](#) Apache 1.2 and up

User tracking using Cookies (replacement for mod_cookies.c)

Miscellaneous

[mod_imap](#) Apache 1.1 and up

The imagemap file handler

[mod_proxy](#) Apache 1.1 and up

Caching proxy abilities

[mod_so](#) Apache 1.3 and up

Support for loading modules (DLLs on Windows) at runtime

[mod_mmap_static](#) Apache 1.3 and up

Experimental file caching, mapping files into memory to improve performance

Development

[mod_example](#) Apache 1.2 and up

Demonstrates Apache API

Obsolete

[mod_browser](#) Apache 1.2.* only

Set environment variables based on User-Agent strings. Replaced by `mod_setenvif` in Apache 1.3 and up

[mod_cookies](#) up to Apache 1.1.1

Support for Netscape-like cookies. Replaced in Apache 1.2 by `mod_usertrack`

[mod_dld](#) Apache 1.2.* and earlier

Start-time linking with the GNU `libdld`. Replaced in Apache 1.3 by `mod_so`

[mod_log_common](#) up to Apache 1.1.1

Standard logging in the Common Logfile Format. Replaced by the `mod_log_config` module in Apache 1.2 and up

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache Directives

Each Apache directive available in the standard Apache distribution is listed here. They are described using a consistent format, and there is [a dictionary](#) of the terms used in their descriptions available.

- [AcceptFilter](#)
- [AcceptMutex](#)
- [AccessConfig](#)
- [AccessFileName](#)
- [Action](#)
- [AddAlt](#)
- [AddAltByEncoding](#)
- [AddAltByType](#)
- [AddCharset](#)
- [AddDefaultCharset](#)
- [AddDescription](#)
- [AddEncoding](#)
- [AddHandler](#)
- [AddIcon](#)
- [AddIconByEncoding](#)
- [AddIconByType](#)
- [AddLanguage](#)
- [AddModule](#)
- [AddModuleInfo](#)
- [AddType](#)
- [AgentLog](#)
- [Alias](#)
- [AliasMatch](#)
- [Allow](#)
- [AllowCONNECT](#)
- [AllowOverride](#)
- [Anonymous](#)
- [Anonymous_Authoritative](#)
- [Anonymous_LogEmail](#)
- [Anonymous_MustGiveEmail](#)

- [Anonymous_NoUserID](#)
- [Anonymous_VerifyEmail](#)
- [AuthAuthoritative](#)
- [AuthDBAuthoritative](#)
- [AuthDBGroupFile](#)
- [AuthDBMAuthoritative](#)
- [AuthDBMGroupFile](#)
- [AuthDBMGroupFile](#)
- [AuthDBUserFile](#)
- [AuthDBMUserFile](#)
- [AuthDigestFile](#)
- [AuthGroupFile](#)
- [AuthName](#)
- [AuthType](#)
- [AuthUserFile](#)
- [BindAddress](#)
- [BrowserMatch](#)
- [BrowserMatchNoCase](#)
- [BS2000Account](#)
- [CacheDefaultExpire](#)
- [CacheDirLength](#)
- [CacheDirLevels](#)
- [CacheForceCompletion](#)
- [CacheGcInterval](#)
- [CacheLastModifiedFactor](#)
- [CacheMaxExpire](#)
- [CacheNegotiatedDocs](#)
- [CacheRoot](#)
- [CacheSize](#)
- [CheckSpelling](#)
- [ClearModuleList](#)
- [ContentDigest](#)
- [CookieExpires](#)
- [CookieLog](#) (mod_cookies)
- [CookieLog](#) (mod_log_config)
- [CookieTracking](#)
- [CoreDumpDirectory](#)
- [CustomLog](#)
- [DefaultIcon](#)

Apache directives

- [DefaultLanguage](#)
- [DefaultType](#)
- [Deny](#)
- [<Directory>](#)
- [<DirectoryMatch>](#)
- [DirectoryIndex](#)
- [DocumentRoot](#)
- [EBCDICConvert](#)
- [EBCDICConvertByType](#)
- [EBCDICKludge](#)
- [ErrorDocument](#)
- [ErrorLog](#)
- [Example](#)
- [ExpiresActive](#)
- [ExpiresByType](#)
- [ExpiresDefault](#)
- [ExtendedStatus](#)
- [FancyIndexing](#)
- [<Files>](#)
- [<FilesMatch>](#)
- [ForceType](#)
- [Group](#)
- [Header](#)
- [HeaderName](#)
- [HostNameLookups](#)
- [IdentityCheck](#)
- [<IfDefine>](#)
- [<IfModule>](#)
- [ImapBase](#)
- [ImapDefault](#)
- [ImapMenu](#)
- [Include](#)
- [IndexIgnore](#)
- [IndexOptions](#)
- [IndexOrderDefault](#)
- [ISAPIReadAheadBuffer](#)
- [ISAPILogNotSupported](#)
- [ISAPIAppendLogToErrors](#)
- [ISAPIAppendLogToQuery](#)

Apache directives

- [KeepAlive](#)
- [KeepAliveTimeout](#)
- [LanguagePriority](#)
- [<Limit>](#)
- [<LimitExcept>](#)
- [LimitRequestBody](#)
- [LimitRequestFields](#)
- [LimitRequestFieldsize](#)
- [LimitRequestLine](#)
- [Listen](#)
- [ListenBacklog](#)
- [LoadFile](#)
- [LoadModule](#)
- [<Location>](#)
- [<LocationMatch>](#)
- [LockFile](#)
- [LogFormat](#)
- [LogLevel](#)
- [MaxClients](#)
- [MaxKeepAliveRequests](#)
- [MaxRequestsPerChild](#)
- [MaxSpareServers](#)
- [MetaDir](#)
- [MetaFiles](#)
- [MetaSuffix](#)
- [MimeMagicFile](#)
- [MinSpareServers](#)
- [MMapFile](#)
- [NameVirtualHost](#)
- [NoCache](#)
- [Options](#)
- [Order](#)
- [PassEnv](#)
- [PidFile](#)
- [Port](#)
- [ProxyBlock](#)
- [ProxyDomain](#)
- [ProxyPass](#)
- [ProxyPassReverse](#)

Apache directives

- [ProxyReceiveBufferSize](#)
- [ProxyRemote](#)
- [ProxyRequests](#)
- [ProxyVia](#)
- [ReadmeName](#)
- [Redirect](#)
- [RedirectMatch](#)
- [RedirectPermanent](#)
- [RedirectTemp](#)
- [RefererIgnore](#)
- [RefererLog](#)
- [RemoveEncoding](#)
- [RemoveHandler](#)
- [RemoveType](#)
- [Require](#)
- [ResourceConfig](#)
- [RewriteBase](#)
- [RewriteCond](#)
- [RewriteEngine](#)
- [RewriteLock](#)
- [RewriteLog](#)
- [RewriteLogLevel](#)
- [RewriteMap](#)
- [RewriteOptions](#)
- [RewriteRule](#)
- [RLimitCPU](#)
- [RLimitMEM](#)
- [RLimitNPROC](#)
- [Satisfy](#)
- [ScoreBoardFile](#)
- [Script](#)
- [ScriptAlias](#)
- [ScriptAliasMatch](#)
- [ScriptInterpreterSource](#)
- [ScriptLog](#)
- [ScriptLogBuffer](#)
- [ScriptLogLength](#)
- [SendBufferSize](#)
- [ServerAdmin](#)

Apache directives

- [ServerAlias](#)
- [ServerName](#)
- [ServerPath](#)
- [ServerRoot](#)
- [ServerSignature](#)
- [ServerTokens](#)
- [ServerType](#)
- [SetEnv](#)
- [SetEnvIf](#)
- [SetEnvIfNoCase](#)
- [SetHandler](#)
- [StartServers](#)
- [ThreadsPerChild](#)
- [TimeOut](#)
- [TransferLog](#)
- [TypesConfig](#)
- [UnsetEnv](#)
- [UseCanonicalName](#)
- [User](#)
- [UserDir](#)
- [<VirtualHost>](#)
- [VirtualDocumentRoot](#)
- [VirtualDocumentRootIP](#)
- [VirtualScriptAlias](#)
- [VirtualScriptAliasIP](#)
- [XBitHack](#)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Terms Used to Describe Apache Directives

Each Apache configuration directive is described using a common format that looks like this:

Syntax: *directive-name some args*

Default: *directive-name default-value*

Context: *context-list*

Override: *override*

Status: *status*

Module: *module-name*

Compatibility: *compatibility notes*

Each of the directive's attributes, complete with possible values where possible, are described in this document.

Directive Terms

- [Syntax](#)
- [Default](#)
- [Context](#)
- [Override](#)
- [Status](#)
- [Module](#)
- [Compatibility](#)

Syntax

This indicates the format of the directive as it would appear in a configuration file. This syntax is extremely directive-specific, and is described in detail in the directive's definition. Generally, the directive name is followed by a series of one or more space-separated arguments. If an argument contains a space, the argument must be enclosed in double quotes. Optional arguments are enclosed in square brackets. Where an argument can take on more than one possible value, the possible values are separated by vertical bars "|". Literal text is presented in the default font, while argument-types for which substitution is necessary are *emphasized*. Directives which can take a variable number of arguments will end in "..." indicating that the last argument is repeated.

Directives use a great number of different argument types. A few common ones are defined below.

URL

A complete Uniform Resource Locator including a scheme, hostname, and optional pathname as in `http://www.example.com/path/to/file.html`

URL-path

The part of a *url* which follows the scheme and hostname as in `/path/to/file.html`. The *url-path* represents a web-view of a resource, as opposed to a file-system view.

file-path

The path to a file in the local file-system beginning with the root directory as in `/usr/local/apache/htdocs/path/to/file.html`. Unless otherwise specified, a *file-path* which does not begin with a slash will be treated as relative to the [ServerRoot](#).

directory-path

The path to a directory in the local file-system beginning with the root directory as in `/usr/local/apache/htdocs/path/to/`.

filename

The name of a file with no accompanying path information as in `file.html`.

regex

A [regular expression](#), which is a way of describing a pattern to match in text. The directive definition will specify what the *regex* is matching against.

extension

In general, this is the part of the *filename* which follows the last dot. However, Apache recognizes multiple filename extensions, so if a *filename* contains more than one dot, each dot-separated part of the filename following the first dot is an *extension*. For example, the *filename* `file.html.en` contains two extensions: `.html` and `.en`. For Apache directives, you may specify *extensions* with or without the leading dot. In addition, *extensions* are not case sensitive.

MIME-type

A method of describing the format of a file which consists of a major format type and a minor format type, separated by a slash as in `text/html`.

env-variable

The name of an [environment variable](#) defined in the Apache configuration process. Note this is not necessarily the same as an operating system environment variable. See the [environment variable documentation](#) for more details.

Default

If the directive has a default value (*i.e.*, if you omit it from your configuration entirely, the Apache Web server will behave as though you set it to a particular value), it is described here. If there is no default value, this section should say "*None*". Note that the default listed here is not necessarily the same as the value the directive takes in the default `httpd.conf` distributed with the server.

Context

This indicates where in the server's configuration files the directive is legal. It's a comma-separated list of one or more of the following values:

server config

This means that the directive may be used in the server configuration files (*e.g.*, `httpd.conf`, `srn.conf`, and `access.conf`), but **not** within any `<VirtualHost>` or `<Directory>` containers. It is not allowed in `.htaccess` files at all.

virtual host

This context means that the directive may appear inside `<VirtualHost>` containers in the server configuration files.

directory

A directive marked as being valid in this context may be used inside `<Directory>`, `<Location>`, and `<Files>` containers in the server configuration files, subject to the restrictions outlined in [How Directory, Location and Files sections work](#).

.htaccess

If a directive is valid in this context, it means that it can appear inside *per-directory* `.htaccess` files. It may not be processed, though depending upon the [overrides](#) currently active.

The directive is *only* allowed within the designated context; if you try to use it elsewhere, you'll get a configuration error that will either prevent the server from handling requests in that context correctly, or will keep the server from operating at all -- *i.e.*, the server won't even start.

The valid locations for the directive are actually the result of a Boolean OR of all of the listed contexts. In other words, a directive that is marked as being valid in "server config, .htaccess" can be used in the httpd.conf file and in .htaccess files, but not within any <Directory> or <VirtualHost> containers.

Override

This directive attribute indicates which configuration override must be active in order for the directive to be processed when it appears in a .htaccess file. If the directive's [context](#) doesn't permit it to appear in .htaccess files, this attribute should say "*Not applicable*".

Overrides are activated by the [AllowOverride](#) directive, and apply to a particular scope (such as a directory) and all descendants, unless further modified by other AllowOverride directives at lower levels. The documentation for that directive also lists the possible override names available.

Status

This indicates how tightly bound into the Apache Web server the directive is; in other words, you may need to recompile the server with an enhanced set of modules in order to gain access to the directive and its functionality. Possible values for this attribute are:

Core

If a directive is listed as having "Core" status, that means it is part of the innermost portions of the Apache Web server, and is always available.

Base

A directive labeled as having "Base" status is supported by one of the standard Apache modules which is compiled into the server by default, and is therefore normally available unless you've taken steps to remove the module from your configuration.

Extension

A directive with "Extension" status is provided by one of the modules included with the Apache server kit, but the module isn't normally compiled into the server. To enable the directive and its functionality, you will need to change the server build configuration files and re-compile Apache.

Experimental

"Experimental" status indicates that the directive is available as part of the Apache kit, but you're on your own if you try to use it. The directive is being documented for completeness, and is not necessarily supported. The module which provides the directive may or may not be compiled in by default; check the top of the page which describes the directive and its module to see if it remarks on the availability.

Module

This quite simply lists the name of the source module which defines the directive.

Compatibility

If the directive wasn't part of the original Apache version 1 distribution, the version in which it was introduced should be listed here. If the directive has the same name as one from the NCSA HTTPd server, any inconsistencies in behavior between the two should also be mentioned. Otherwise, this attribute should say "*No compatibility issues.*"

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Terms Used to Describe Apache Modules

Each Apache module is described using a common format that looks like this:

Status: *status*

Source File: *source-file*

Module Identifier: *module-identifier*

Compatibility: *compatibility notes*

Each of the attributes, complete with values where possible, are described in this document.

Module Terms

- [Status](#)
 - [Source File](#)
 - [Module Identifier](#)
 - [Compatibility](#)
-

Status

This indicates how tightly bound into the Apache Web server the module is; in other words, you may need to recompile the server in order to gain access to the module and its functionality. Possible values for this attribute are:

Base

A module labeled as having "Base" status is compiled and loaded into the server by default, and is therefore normally available unless you have taken steps to remove the module from your configuration.

Extension

A module with "Extension" status is not normally compiled and loaded into the server. To enable the module and its functionality, you may need to change the server build configuration files and re-compile Apache.

Experimental

"Experimental" status indicates that the module is available as part of the Apache kit, but you are on your own if you try to use it. The module is being documented for completeness, and is not necessarily supported.

External

Modules which are not included with the base Apache distribution ("third-party modules") may use the "External" status. We are not responsible, nor do we support such modules.

Source File

This quite simply lists the name of the source file which contains the code for the module. This is also the name used by the [<IfModule>](#) directive.

Module Identifier

This is a string which identifies the module for use in the [LoadModule](#) directive when dynamically loading modules. In particular, it is the name of the external variable of type module in the source file.

Compatibility

If the module was not part of the original Apache version 1 distribution, the version in which it was introduced should be listed here.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache Core Features

These configuration parameters control the core Apache features, and are always available.

Directives

- [AcceptFilter](#)
- [AcceptMutex](#)
- [AccessConfig](#)
- [AccessFileName](#)
- [AddDefaultCharset](#)
- [AddModule](#)
- [AllowOverride](#)
- [AuthName](#)
- [AuthType](#)
- [BindAddress](#)
- [BS2000Account](#)
- [ClearModuleList](#)
- [ContentDigest](#)
- [CoreDumpDirectory](#)
- [DefaultType](#)
- [<Directory>](#)
- [<DirectoryMatch>](#)
- [DocumentRoot](#)
- [EBCDICConvert](#)
- [EBCDICConvertByType](#)
- [EBCDICKludge](#)
- [ErrorDocument](#)
- [ErrorLog](#)
- [<Files>](#)
- [<FilesMatch>](#)
- [Group](#)
- [HostNameLookups](#)
- [IdentityCheck](#)

- [<IfDefine>](#)
- [<IfModule>](#)
- [Include](#)
- [KeepAlive](#)
- [KeepAliveTimeout](#)
- [<Limit>](#)
- [<LimitExcept>](#)
- [LimitRequestBody](#)
- [LimitRequestFields](#)
- [LimitRequestFieldsize](#)
- [LimitRequestLine](#)
- [Listen](#)
- [ListenBacklog](#)
- [<Location>](#)
- [<LocationMatch>](#)
- [LockFile](#)
- [LogLevel](#)
- [MaxClients](#)
- [MaxKeepAliveRequests](#)
- [MaxRequestsPerChild](#)
- [MaxSpareServers](#)
- [MinSpareServers](#)
- [NameVirtualHost](#)
- [Options](#)
- [PidFile](#)
- [Port](#)
- [Require](#)
- [ResourceConfig](#)
- [RLimitCPU](#)
- [RLimitMEM](#)
- [RLimitNPROC](#)
- [Satisfy](#)
- [ScoreBoardFile](#)
- [ScriptInterpreterSource](#)
- [SendBufferSize](#)
- [ServerAdmin](#)
- [ServerAlias](#)
- [ServerName](#)
- [ServerPath](#)

- [ServerRoot](#)
 - [ServerSignature](#)
 - [ServerTokens](#)
 - [ServerType](#)
 - [StartServers](#)
 - [ThreadsPerChild](#)
 - [ThreadStackSize](#)
 - [TimeOut](#)
 - [UseCanonicalName](#)
 - [User](#)
 - [<VirtualHost>](#)
-

AcceptFilter directive

Syntax: `AcceptFilter on|off`

Default: `AcceptFilter on`

Context: server config

Status: core **Compatibility:** AcceptFilter is available in Apache 1.3.22 and later

AcceptFilter controls a BSD specific filter optimization. It is compiled in by default - and switched on by default if your system supports it (setsocketopt() option SO_ACCEPTFILTER). Currently only FreeBSD supports this.

See the filter section on [performance hints](#) for more information.

The compile time flag `AP_ACCEPTFILTER_OFF` can be used to change the default to 'off'. `httpd -V` and `httpd -L` will show compile time defaults and whether or not `SO_ACCEPTFILTER` was defined during the compile.

AcceptMutex directive

Syntax: `AcceptMutex uslock|pthread|sysvsem|fcntl|flock|os2sem|tpfcore|none|default`

Default: `AcceptMutex default`

Context: server config

Status: core

AcceptMutex controls which accept() mutex method Apache will use. Not all methods are available on all platforms, since the suite of methods is determined at compile-time. For a list of which methods are available for your particular build, the `httpd -L` command line option will list them out.

The compile time flags `-D HAVE_METHOD_SERIALIZED_ACCEPT` can be used to add different methods to your build, or one can edit the `include/ap_config.h` file for your particular platform.

See the [performance tuning guide](#) for more information.

AccessConfig directive

Syntax: `AccessConfig file-path|directory-path`

Default: `AccessConfig conf/access.conf`

Context: server config, virtual host

Status: core

Compatibility: The ability to specify a directory, rather than a file name, is only available in Apache 1.3.13 and later.

The server will read this file for more directives after reading the [ResourceConfig](#) file. *File-path* is relative to the [ServerRoot](#). This feature can be disabled using:

```
AccessConfig /dev/null
```

Or, on Win32 servers,

```
AccessConfig nul
```

Historically, this file only contained [<Directory>](#) sections; in fact it can now contain any server directive allowed in the *server config* context. However, since Apache version 1.3.4, the default `access.conf` file which ships with Apache contains only comments, and all directives are placed in the main server configuration file, `httpd.conf`.

If `AccessConfig` points to a directory, rather than a file, Apache will read all files in that directory, and any subdirectory, and parse those as configuration files. Note that any file in the specified directory will be loaded as a configuration file, so make sure that you don't have stray files in this directory by mistake, such as temporary files created by your editor, for example.

See also [ResourceConfig](#).

AccessFileName directive

Syntax: `AccessFileName filename [filename] ...`

Default: `AccessFileName .htaccess`

Context: server config, virtual host

Status: core

Compatibility: `AccessFileName` can accept more than one filename only in Apache 1.3 and later

When returning a document to the client the server looks for the first existing access control file from this list of names in every directory of the path to the document, if access control files are enabled for that directory. For example:

```
AccessFileName .acl
```

before returning the document `/usr/local/web/index.html`, the server will read `/.acl`, `/usr/.acl`, `/usr/local/.acl` and `/usr/local/web/.acl` for directives, unless they have been disabled with

```
<Directory />
AllowOverride None
</Directory>
```

See Also: [AllowOverride](#)

AddDefaultCharset directive

Syntax: `AddDefaultCharset On|Off|charset`

Context: all

Status: core

Default: `AddDefaultCharset Off`

Compatibility: `AddDefaultCharset` is only available in Apache 1.3.12 and later

This directive specifies the name of the character set that will be added to any response that does not have any parameter on the content type in the HTTP headers. This will override any character set specified in the body of the document via a `META` tag. A setting of `AddDefaultCharset Off` disables this functionality. `AddDefaultCharset On` enables Apache's internal default charset of `iso-8859-1` as required by the directive. You can also specify an alternate *charset* to be used; e.g.

```
AddDefaultCharset utf-8.
```

AddModule directive

Syntax: AddModule *module* [*module*] ...

Context: server config

Status: core

Compatibility: AddModule is only available in Apache 1.2 and later

The server can have modules compiled in which are not actively in use. This directive can be used to enable the use of those modules. The server comes with a pre-loaded list of active modules; this list can be cleared with the [ClearModuleList](#) directive.

AllowOverride directive

Syntax: AllowOverride All|None|*directive-type* [*directive-type*] ...

Default: AllowOverride All

Context: directory

Status: core

When the server finds an .htaccess file (as specified by [AccessFileName](#)) it needs to know which directives declared in that file can override earlier access information.

When this directive is set to None, then .htaccess files are completely ignored. In this case, the server will not even attempt to read .htaccess files in the filesystem.

When this directive is set to All, then any directive which has the .htaccess [Context](#) is allowed in .htaccess files.

The *directive-type* can be one of the following groupings of directives.

AuthConfig

Allow use of the authorization directives ([AuthDBMGroupFile](#), [AuthDBMUserFile](#), [AuthGroupFile](#), [AuthName](#), [AuthType](#), [AuthUserFile](#), [Require](#), *etc.*).

FileInfo

Allow use of the directives controlling document types ([AddEncoding](#), [AddLanguage](#), [AddType](#), [DefaultType](#), [ErrorDocument](#), [LanguagePriority](#), *etc.*).

Indexes

Allow use of the directives controlling directory indexing ([AddDescription](#), [AddIcon](#), [AddIconByEncoding](#), [AddIconByType](#), [DefaultIcon](#), [DirectoryIndex](#), [FancyIndexing](#), [HeaderName](#), [IndexIgnore](#), [IndexOptions](#), [ReadmeName](#), *etc.*).

Limit

Allow use of the directives controlling host access (Allow, Deny and Order).

Options

Allow use of the directives controlling specific directory features ([Options](#) and [XBitHack](#)).

See Also: [AccessFileName](#)

AuthName directive

Syntax: AuthName *auth-domain*

Context: directory, .htaccess

Override: AuthConfig

Status: core

This directive sets the name of the authorization realm for a directory. This realm is given to the client so that the user knows which username and password to send. `AuthName` takes a single argument; if the realm name contains spaces, it must be enclosed in quotation marks. It must be accompanied by [AuthType](#) and [Require](#) directives, and directives such as [AuthUserFile](#) and [AuthGroupFile](#) to work.

AuthType directive

Syntax: `AuthType Basic|Digest`

Context: directory, `.htaccess`

Override: `AuthConfig`

Status: core

This directive selects the type of user authentication for a directory. Only `Basic` and `Digest` are currently implemented. It must be accompanied by [AuthName](#) and [Require](#) directives, and directives such as [AuthUserFile](#) and [AuthGroupFile](#) to work.

BindAddress directive

Syntax: `BindAddress *|IP-address|domain-name`

Default: `BindAddress *`

Context: server config

Status: core

A Unix® http server can either listen for connections to every IP address of the server machine, or just one IP address of the server machine. If the argument to this directive is `*`, then the server will listen for connections on every IP address. Otherwise, the server can listen to only a specific *IP-address* or a fully-qualified Internet *domain-name*.

Only one `BindAddress` directive can be used. For more control over which address and ports Apache listens to, use the [Listen](#) directive instead of `BindAddress`.

`BindAddress` can be used as an alternative method for supporting [virtual hosts](#) using multiple independent servers, instead of using `<VirtualHost>` sections.

See Also: [DNS Issues](#)

See Also: [Setting which addresses and ports Apache uses](#)

BS2000Account directive

Syntax: `BS2000Account account`

Default: `none`

Context: server config

Status: core

Compatibility: `BS2000Account` is only available for BS2000 machines, as of Apache 1.3 and later.

The `BS2000Account` directive is available for BS2000 hosts only. It must be used to define the account number for the non-privileged apache server user (which was configured using the [User](#) directive). This is required by the BS2000 POSIX subsystem (to change the underlying BS2000 task environment by performing a sub-LOGON) to prevent CGI scripts from accessing resources of the privileged account which started the server, usually `SYSROOT`.

Only one `BS2000Account` directive can be used.

See Also: [Apache EBCDIC port](#)

ClearModuleList directive

Syntax: ClearModuleList

Context: server config

Status: core

Compatibility: ClearModuleList is only available in Apache 1.2 and later

The server comes with a built-in list of active modules. This directive clears the list. It is assumed that the list will then be re-populated using the [AddModule](#) directive.

ContentDigest directive

Syntax: ContentDigest on|off

Default: ContentDigest off

Context: server config, virtual host, directory, .htaccess

Override: Options

Status: experimental

Compatibility: ContentDigest is only available in Apache 1.1 and later

This directive enables the generation of `Content-MD5` headers as defined in RFC1864 respectively RFC2068.

MD5 is an algorithm for computing a "message digest" (sometimes called "fingerprint") of arbitrary-length data, with a high degree of confidence that any alterations in the data will be reflected in alterations in the message digest.

The `Content-MD5` header provides an end-to-end message integrity check (MIC) of the entity-body. A proxy or client may check this header for detecting accidental modification of the entity-body in transit. Example header:

```
Content-MD5: AuLb7Dp1rqtRtxz2m9kRpA==
```

Note that this can cause performance problems on your server since the message digest is computed on every request (the values are not cached).

`Content-MD5` is only sent for documents served by the core, and not by any module. For example, SSI documents, output from CGI scripts, and byte range responses do not have this header.

CoreDumpDirectory directive

Syntax: CoreDumpDirectory *directory-path*

Default: the same location as ServerRoot

Context: server config

Status: core

This controls the directory to which Apache attempts to switch before dumping core. The default is in the [ServerRoot](#) directory, however since this should not be writable by the user the server runs as, core dumps won't normally get written. If you want a core dump for debugging, you can use this directive to place it in a different location.

DefaultType directive

Syntax: DefaultType *MIME-type*

Default: DefaultType text/html

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: core

There will be times when the server is asked to provide a document whose type cannot be determined by its MIME types mappings.

The server must inform the client of the content-type of the document, so in the event of an unknown type it uses the DefaultType. For example:

```
DefaultType image/gif
```

would be appropriate for a directory which contained many gif images with filenames missing the .gif extension.

<Directory> directive

Syntax: <Directory *directory-path*> ... </Directory>

Context: server config, virtual host

Status: Core.

<Directory> and </Directory> are used to enclose a group of directives which will apply only to the named directory and sub-directories of that directory. Any directive which is allowed in a directory context may be used. *Directory-path* is either the full path to a directory, or a wild-card string. In a wild-card string, '?' matches any single character, and '*' matches any sequences of characters. As of Apache 1.3, you may also use '[' character ranges like in the shell. Also as of Apache 1.3 none of the wildcards match a '/' character, which more closely mimics the behavior of Unix shells. Example:

```
<Directory /usr/local/httpd/htdocs>
Options Indexes FollowSymLinks
</Directory>
```

Apache 1.2 and above: Extended regular expressions can also be used, with the addition of the ~ character. For example:

```
<Directory ~ "^/www/.*/[0-9]{3}">
```

would match directories in /www/ that consisted of three numbers.

If multiple (non-regular expression) directory sections match the directory (or its parents) containing a document, then the directives are applied in the order of shortest match first, interspersed with the directives from the [.htaccess](#) files. For example, with

```
<Directory />
AllowOverride None
</Directory>

<Directory /home/*>
AllowOverride FileInfo
</Directory>
```

for access to the document /home/web/dir/doc.html the steps are:

- Apply directive AllowOverride None (disabling .htaccess files).
- Apply directive AllowOverride FileInfo (for directory /home/web).
- Apply any FileInfo directives in /home/web/.htaccess

Regular expression directory sections are handled slightly differently by Apache 1.2 and 1.3. In Apache 1.2 they are interspersed with the normal directory sections and applied in the order they appear in the configuration file. They are applied only once, and apply when the shortest match possible occurs. In Apache 1.3 regular expressions are not considered until after all of the normal sections have been applied. Then all of the regular expressions are tested in the order they appeared in the configuration file. For

example, with

```
<Directory ~ abc$>
... directives here ...
</Directory>
```

Suppose that the filename being accessed is `/home/abc/public_html/abc/index.html`. The server considers each of `/`, `/home`, `/home/abc`, `/home/abc/public_html`, and `/home/abc/public_html/abc` in that order. In Apache 1.2, when `/home/abc` is considered, the regular expression will match and be applied. In Apache 1.3 the regular expression isn't considered at all at that point in the tree. It won't be considered until after all normal `<Directory>`s and `.htaccess` files have been applied. Then the regular expression will match on `/home/abc/public_html/abc` and be applied.

Note that the default Apache access for `<Directory />` is `Allow from All`. This means that Apache will serve any file mapped from an URL. It is recommended that you change this with a block such as

```
<Directory />
    Order Deny,Allow
    Deny from All
</Directory>
```

and then override this for directories you *want* accessible. See the [Security Tips](#) page for more details.

The directory sections typically occur in the `access.conf` file, but they may appear in any configuration file. `<Directory>` directives cannot nest, and cannot appear in a `<Limit>` or `<LimitExcept>` section.

See also: [How Directory, Location and Files sections work](#) for an explanation of how these different sections are combined when a request is received

<DirectoryMatch>

Syntax: `<DirectoryMatch regex> ... </DirectoryMatch>`

Context: server config, virtual host

Status: Core.

Compatibility: Available in Apache 1.3 and later

`<DirectoryMatch>` and `</DirectoryMatch>` are used to enclose a group of directives which will apply only to the named directory and sub-directories of that directory, the same as `<Directory>`. However, it takes as an argument a regular expression. For example:

```
<DirectoryMatch "^/www/.*/[0-9]{3}">
```

would match directories in `/www/` that consisted of three numbers.

See Also: `<Directory>` for a description of how regular expressions are mixed in with normal `<Directory>`s.

See also: [How Directory, Location and Files sections work](#) for an explanation of how these different sections are combined when a request is received

DocumentRoot directive

Syntax: `DocumentRoot directory-path`

Default: `DocumentRoot /usr/local/apache/htdocs`

Context: server config, virtual host

Status: core

This directive sets the directory from which `httpd` will serve files. Unless matched by a directive like `Alias`, the server appends the path from the requested URL to the document root to make the path to the document. Example:

```
DocumentRoot /usr/web
```

then an access to `http://www.my.host.com/index.html` refers to `/usr/web/index.html`.

There appears to be a bug in `mod_dir` which causes problems when the `DocumentRoot` has a trailing slash (*i.e.*, "`DocumentRoot /usr/web/`") so please avoid that.

EBCDICConvert

Syntax: `EBCDICConvert On|Off[=direction] extension [extension] ...`

Context: server config, virtual host, directory, `.htaccess`

Status: core

Override: `FileInfo`

Compatibility: The configurable EBCDIC conversion is only available in Apache 1.3.19 and later, and on EBCDIC based platforms.

The `EBCDICConvert` directive maps the given filename extensions to the specified conversion setting (On or Off). File extensions may be specified with or without a leading dot.

If the optional format `On=direction` (or `Off=direction`) is used, where *direction* is one of In, Out or InOut, then the directive only applies to the specified transfer direction (In: uploaded content in a PUT or POST request, Out: returned content in a GET or POST request, and InOut: conversion in both directions).

Otherwise, InOut (conversion in both directions) is implied.

Conversion configuration based on file extension is tested prior to configuration based on MIME type, to allow for generic MIME based rules to be overridden by a more specific file extension (several file extensions may exist for the same MIME type).

Example:

With a configuration like the following, the normal `*.html` files contain HTML text in EBCDIC encoding, while `*.ahtml` files contain HTML text in ASCII encoding:

```
# *.html and *.ahtml contain HTML text:
AddType text/html .html .ahtml

# *.ahtml is not converted (contains ASCII text already):
EBCDICConvert Off .ahtml

# All other text/html files presumably contain EBCDIC text:
EBCDICConvertByType On text/html
```

See also: [EBCDICConvertByType](#) and [Overview of the EBCDIC Conversion Functions](#)

EBCDICConvertByType

Syntax: `EBCDICConvertByType On|Off[=direction] mimetype [mimetype] ...`

Context: server config, virtual host, directory, `.htaccess`

Status: core

Override: `FileInfo`

Compatibility: The configurable EBCDIC conversion is only available in Apache 1.3.19 and later, and on EBCDIC based platforms.

The `EBCDICConvertByType` directive maps the given MIME type (optionally containing wildcards) to the specified conversion setting (On or Off).

If the optional format `On=direction` (or `Off=direction`) is used, where *direction* is one of In, Out or InOut, then the directive only

applies to the specified transfer direction (In: uploaded content in a PUT or POST request, Out: returned content in a GET or POST request, and InOut: conversion in both directions).

Otherwise, InOut (conversion in both directions) is implied.

Example:

A useful standard configuration should at least contain the following defaults:

```
# All text documents are stored as EBCDIC files:
EBCDICConvertByType On  text/* message/* multipart/*
EBCDICConvertByType On  application/x-www-form-urlencoded \
                        model/vrml application/postscript
# All other files are assumed to be binary:
EBCDICConvertByType Off */*
```

If you serve ASCII documents only, for example from an NFS mounted unix server, use:

```
# All documents are ASCII already:
EBCDICConvertByType Off */*
```

See also: [EBCDICConvert](#) and [Overview of the EBCDIC Conversion Functions](#)

EBCDICKludge

Syntax: EBCDICKludge On|Off

Default: EBCDICKludge Off

Context: server config, virtual host, directory, .htaccess

Status: core

Override: FileInfo

Compatibility: EBCDICKludge is only available in Apache 1.3.19 and later, and on EBCDIC based platforms. It is deprecated and will be withdrawn in a future version.

The EBCDICKludge is provided for the backward compatible behavior with apache versions 1.3.0 through 1.3.18. In these versions, all files with MIME types starting with "text/", "message/" or "multipart/" or with type "application/x-www-form-urlencoded" would be converted by default, all other documents were returned unconverted. Only if a MIME type "text/x-ascii-subtype" was configured for a certain document, the document was assumed to be in ASCII format already, and was not converted again. Instead, the "x-ascii-" was removed from the type, resulting in the MIME type "text/subtype" being returned for the document.

If the EBCDICKludge directive is set to On, and if none of the file extensions configured with the [EBCDICConvert](#) directive matches in the current context, then the server tests for a MIME type of the format *type/x-ascii-subtype*. If the document has such a type, then the "x-ascii-" substring is removed and the conversion set to Off. This allows for overriding the implicit assumption that all text files are stored in EBCDIC format, for example when serving documents from an NFS mounted directory with ASCII documents.

By using the EBCDICKludge, there is no way to force one of the other MIME types (*e.g.*, model/vrml) to be treated as an EBCDIC text file. Use of the [EBCDICConvertByType](#) directive mentioned above is the preferred way to configure such a conversion. (Before Apache version 1.3.19, there was no way at all to force these binary documents to be treated as EBCDIC text files.)

See also: [EBCDICConvert](#), [EBCDICConvertByType](#) and [Overview of the EBCDIC Conversion Functions](#)

ErrorDocument directive

Syntax: ErrorDocument *error-code document*

Context: server config, virtual host, directory, .htaccess

Status: core

Override: FileInfo

Compatibility: The directory and .htaccess contexts are only available in Apache 1.1 and later.

In the event of a problem or error, Apache can be configured to do one of four things,

1. output a simple hardcoded error message
2. output a customized message
3. redirect to a local *URL-path* to handle the problem/error
4. redirect to an external *URL* to handle the problem/error

The first option is the default, while options 2-4 are configured using the `ErrorDocument` directive, which is followed by the HTTP response code and a message or URL.

Messages in this context begin with a single double-quote character (`"`), which does not form part of the message itself. Apache will sometimes offer additional information regarding the problem/error.

URLs can begin with a slash (`/`) for local URLs, or be a full URL which the client can resolve. Examples:

```
ErrorDocument 500 http://foo.example.com/cgi-bin/tester
ErrorDocument 404 /cgi-bin/bad_urls.pl
ErrorDocument 401 /subscription_info.html
ErrorDocument 403 "Sorry can't allow you access today"
```

Note that when you specify an `ErrorDocument` that points to a remote URL (ie. anything with a method such as "http" in front of it), Apache will send a redirect to the client to tell it where to find the document, even if the document ends up being on the same server. This has several implications, the most important being that the client will not receive the original error status code, but instead will receive a redirect status code. This in turn can confuse web robots and other clients which try to determine if a URL is valid using the status code. In addition, if you use a remote URL in an `ErrorDocument 401`, the client will not know to prompt the user for a password since it will not receive the 401 status code. Therefore, **if you use an "ErrorDocument 401" directive then it must refer to a local document.**

See Also: [documentation of customizable responses](#).

ErrorLog directive

Syntax: `ErrorLog file-path|syslog[:facility]`

Default: `ErrorLog logs/error_log` (Unix)

Default: `ErrorLog logs/error.log` (Windows and OS/2)

Context: server config, virtual host

Status: core

The error log directive sets the name of the file to which the server will log any errors it encounters. If the *file-path* does not begin with a slash (`/`) then it is assumed to be relative to the [ServerRoot](#). If the *file-path* begins with a pipe (`|`) then it is assumed to be a command to spawn to handle the error log.

Apache 1.3 and above: Using `syslog` instead of a filename enables logging via `syslogd(8)` if the system supports it. The default is to use `syslog` facility `local7`, but you can override this by using the `syslog:facility` syntax where *facility* can be one of the names usually documented in `syslog(1)`.

SECURITY: See the [security tips](#) document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

See also: [LogLevel](#) and [Apache Log Files](#)

<Files> directive

Syntax: <Files *filename*> ... </Files>

Context: server config, virtual host, .htaccess

Status: core

Compatibility: only available in Apache 1.2 and above.

The <Files> directive provides for access control by filename. It is comparable to the [<Directory>](#) directive and [<Location>](#) directives. It should be matched with a </Files> directive. The directives given within this section will be applied to any object with a basename (last component of filename) matching the specified filename. <Files> sections are processed in the order they appear in the configuration file, after the <Directory> sections and .htaccess files are read, but before <Location> sections. Note that <Files> can be nested inside <Directory> sections to restrict the portion of the filesystem they apply to.

The *filename* argument should include a filename, or a wild-card string, where '?' matches any single character, and '*' matches any sequences of characters. Extended regular expressions can also be used, with the addition of the '~' character. For example:

```
<Files ~ "\.(gif|jpe?g|png)$">
```

would match most common Internet graphics formats. In Apache 1.3 and later, [<FilesMatch>](#) is preferred, however.

Note that unlike [<Directory>](#) and [<Location>](#) sections, <Files> sections can be used inside .htaccess files. This allows users to control access to their own files, at a file-by-file level.

See also: [How Directory, Location and Files sections work](#) for an explanation of how these different sections are combined when a request is received

<FilesMatch>

Syntax: <FilesMatch *regex*> ... </FilesMatch>

Context: server config, virtual host, .htaccess

Status: core

Compatibility: only available in Apache 1.3 and above.

The <FilesMatch> directive provides for access control by filename, just as the [<Files>](#) directive does. However, it accepts a regular expression. For example:

```
<FilesMatch "\.(gif|jpe?g|png)$">
```

would match most common Internet graphics formats.

See also: [How Directory, Location and Files sections work](#) for an explanation of how these different sections are combined when a request is received

Group directive

Syntax: Group *unix-group*

Default: Group #-1

Context: server config, virtual host

Status: core

The Group directive sets the group under which the server will answer requests. In order to use this directive, the stand-alone server must be run initially as root. *Unix-group* is one of:

A group name

Refers to the given group by name.

followed by a group number.

Refers to a group by its number.

It is recommended that you set up a new group specifically for running the server. Some admins use user `nobody`, but this is not always possible or desirable.

Note: if you start the server as a non-root user, it will fail to change to the specified group, and will instead continue to run as the group of the original user.

Special note: Use of this directive in `<VirtualHost>` requires a properly configured [suEXEC wrapper](#). When used inside a `<VirtualHost>` in this manner, only the group that CGIs are run as is affected. Non-CGI requests are still processed as the group specified in the main `Group` directive.

SECURITY: See [User](#) for a discussion of the security considerations.

HostNameLookups directive

Syntax: `HostNameLookups on|off|double`

Default: `HostNameLookups off`

Context: server config, virtual host, directory

Status: core

Compatibility: `double` available only in Apache 1.3 and above.

Compatibility: Default was `on` prior to Apache 1.3.

This directive enables DNS lookups so that host names can be logged (and passed to CGIs/SSIs in `REMOTE_HOST`). The value `double` refers to doing double-reverse DNS. That is, after a reverse lookup is performed, a forward lookup is then performed on that result. At least one of the ip addresses in the forward lookup must match the original address. (In "tcpwrappers" terminology this is called `PARANOID`.)

Regardless of the setting, when [mod_access](#) is used for controlling access by hostname, a double reverse lookup will be performed. This is necessary for security. Note that the result of this double-reverse isn't generally available unless you set `HostnameLookups double`. For example, if only `HostnameLookups on` and a request is made to an object that is protected by hostname restrictions, regardless of whether the double-reverse fails or not, CGIs will still be passed the single-reverse result in `REMOTE_HOST`.

The default for this directive was previously `on` in versions of Apache prior to 1.3. It was changed to `off` in order to save the network traffic for those sites that don't truly need the reverse lookups done. It is also better for the end users because they don't have to suffer the extra latency that a lookup entails. Heavily loaded sites should leave this directive `off`, since DNS lookups can take considerable amounts of time. The utility [logresolve](#), provided in the `/support` directory, can be used to look up host names from logged IP addresses offline.

IdentityCheck directive

Syntax: `IdentityCheck on|off`

Default: `IdentityCheck off`

Context: server config, virtual host, directory

Status: core

This directive enables RFC1413-compliant logging of the remote user name for each connection, where the client machine runs `identd` or something similar. This information is logged in the access log. *Boolean* is either `on` or `off`.

The information should not be trusted in any way except for rudimentary usage tracking.

Note that this can cause serious latency problems accessing your server since every request requires one of these lookups to be

performed. When firewalls are involved each lookup might possibly fail and add 30 seconds of latency to each hit. So in general this is not very useful on public servers accessible from the Internet.

<IfDefine> directive

Syntax: <IfDefine [!]*parameter-name*> ... </IfDefine>

Default: None

Context: all

Status: Core

Compatibility: <IfDefine> is only available in 1.3.1 and later.

The <IfDefine *test*>...</IfDefine> section is used to mark directives that are conditional. The directives within an IfDefine section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

The *test* in the <IfDefine> section directive can be one of two forms:

- *parameter-name*
- *!parameter-name*

In the former case, the directives between the start and end markers are only processed if the parameter named *parameter-name* is defined. The second format reverses the test, and only processes the directives if *parameter-name* is **not** defined.

The *parameter-name* argument is a define as given on the httpd command line via `-Dparameter-`, at the time the server was started.

<IfDefine> sections are nest-able, which can be used to implement simple multiple-parameter tests. Example:

```
$ httpd -DReverseProxy ...

# httpd.conf
<IfDefine ReverseProxy>
LoadModule rewrite_module libexec/mod_rewrite.so
LoadModule proxy_module    libexec/libproxy.so
</IfDefine>
```

<IfModule> directive

Syntax: <IfModule [!]*module-name*> ... </IfModule>

Default: None

Context: all

Status: Core

Compatibility: IfModule is only available in 1.2 and later.

The <IfModule *test*>...</IfModule> section is used to mark directives that are conditional. The directives within an IfModule section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

The *test* in the <IfModule> section directive can be one of two forms:

- *module name*
- *!module name*

In the former case, the directives between the start and end markers are only processed if the module named *module name* is compiled in to Apache. The second format reverses the test, and only processes the directives if *module name* is **not** compiled in.

The *module name* argument is a module name as given as the file name of the module, at the time it was compiled. For example, `mod_rewrite.c`.

<IfModule> sections are nest-able, which can be used to implement simple multiple-module tests.

Include directive

Syntax: Include *file-path|directory-path*

Context: server config

Status: Core

Compatibility: Include is only available in Apache 1.3 and later.

This directive allows inclusion of other configuration files from within the server configuration files.

New in Apache 1.3.13 is the feature that if `Incl` points to a directory, rather than a file, Apache will read all files in that directory, and any subdirectory, and parse those as configuration files.

KeepAlive directive

Syntax: (Apache 1.1) KeepAlive *max-requests*

Default: (Apache 1.1) KeepAlive 5

Syntax: (Apache 1.2) KeepAlive on|off

Default: (Apache 1.2) KeepAlive On

Context: server config

Status: Core

Compatibility: KeepAlive is only available in Apache 1.1 and later.

The Keep-Alive extension to HTTP/1.0 and the persistent connection feature of HTTP/1.1 provide long-lived HTTP sessions which allow multiple requests to be sent over the same TCP connection. In some cases this has been shown to result in an almost 50% speedup in latency times for HTML documents with many images. To enable Keep-Alive connections in Apache 1.2 and later, set `KeepAlive On`.

For HTTP/1.0 clients, Keep-Alive connections will only be used if they are specifically requested by a client. In addition, a Keep-Alive connection with an HTTP/1.0 client can only be used when the length of the content is known in advance. This implies that dynamic content such as CGI output, SSI pages, and server-generated directory listings will generally not use Keep-Alive connections to HTTP/1.0 clients. For HTTP/1.1 clients, persistent connections are the default unless otherwise specified. If the client requests it, chunked encoding will be used in order to send content of unknown length over persistent connections.

Apache 1.1 only: Set *max-requests* to the maximum number of requests you want Apache to entertain per connection. A limit is imposed to prevent a client from hogging your server resources. Set this to 0 to disable support. In Apache 1.2 and 1.3, this is controlled through the `MaxKeepAliveRequests` directive instead.

See also [MaxKeepAliveRequests](#).

KeepAliveTimeout directive

Syntax: KeepAliveTimeout *seconds*

Default: KeepAliveTimeout 15

Context: server config

Status: Core

Compatibility: KeepAliveTimeout is only available in Apache 1.1 and later.

The number of seconds Apache will wait for a subsequent request before closing the connection. Once a request has been received, the timeout value specified by the [Timeout](#) directive applies.

Setting `KeepAliveTimeout` to a high value may cause performance problems in heavily loaded servers. The higher the timeout, the more server processes will be kept occupied waiting on connections with idle clients.

<Limit> directive

Syntax: `<Limit method [method] ... > ... </Limit>`

Context: any

Status: core

Access controls are normally effective for **all** access methods, and this is the usual desired behavior. **In the general case, access control directives should not be placed within a <limit> section.**

The purpose of the <Limit> directive is to restrict the effect of the access controls to the nominated HTTP methods. For all other methods, the access restrictions that are enclosed in the <Limit> bracket **will have no effect**. The following example applies the access control only to the methods POST, PUT, and DELETE, leaving all other methods unprotected:

```
<Limit POST PUT DELETE>
Require valid-user
</Limit>
```

The method names listed can be one or more of: GET, POST, PUT, DELETE, CONNECT, OPTIONS, TRACE, PATCH, PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, and UNLOCK. **The method name is case-sensitive.** If GET is used it will also restrict HEAD requests.

<LimitExcept> directive

Syntax: `<LimitExcept method [method] ... > ... </LimitExcept>`

Context: any

Status: core

Compatibility: Available in Apache 1.3.5 and later

<LimitExcept> and </LimitExcept> are used to enclose a group of access control directives which will then apply to any HTTP access method **not** listed in the arguments; *i.e.*, it is the opposite of a <Limit> section and can be used to control both standard and nonstandard/unrecognized methods. See the documentation for <Limit> for more details.

LimitRequestBody directive

Syntax: `LimitRequestBody bytes`

Default: `LimitRequestBody 0`

Context: server config, virtual host, directory, .htaccess

Status: core

Compatibility: `LimitRequestBody` is only available in Apache 1.3.2 and later.

This directive specifies the number of *bytes* from 0 (meaning unlimited) to 2147483647 (2GB) that are allowed in a request body. The default value is defined by the compile-time constant `DEFAULT_LIMIT_REQUEST_BODY` (0 as distributed).

The `LimitRequestBody` directive allows the user to set a limit on the allowed size of an HTTP request message body within the context in which the directive is given (server, per-directory, per-file or per-location). If the client request exceeds that limit, the server will return an error response instead of servicing the request. The size of a normal request message body will vary greatly depending on the nature of the resource and the methods allowed on that resource. CGI scripts typically use the message body for passing form information to the server. Implementations of the PUT method will require a value at least as large as any representation that the server wishes to accept for that resource.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for

avoiding some forms of denial-of-service attacks.

LimitRequestFields directive

Syntax: LimitRequestFields *number*

Default: LimitRequestFields 100

Context: server config

Status: core

Compatibility: LimitRequestFields is only available in Apache 1.3.2 and later.

Number is an integer from 0 (meaning unlimited) to 32767. The default value is defined by the compile-time constant `DEFAULT_LIMIT_REQUEST_FIELDS` (100 as distributed).

The LimitRequestFields directive allows the server administrator to modify the limit on the number of request header fields allowed in an HTTP request. A server needs this value to be larger than the number of fields that a normal client request might include. The number of request header fields used by a client rarely exceeds 20, but this may vary among different client implementations, often depending upon the extent to which a user has configured their browser to support detailed content negotiation. Optional HTTP extensions are often expressed using request header fields.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks. The value should be increased if normal clients see an error response from the server that indicates too many fields were sent in the request.

LimitRequestFieldsize directive

Syntax: LimitRequestFieldsize *bytes*

Default: LimitRequestFieldsize 8190

Context: server config

Status: core

Compatibility: LimitRequestFieldsize is only available in Apache 1.3.2 and later.

This directive specifies the number of *bytes* from 0 to the value of the compile-time constant `DEFAULT_LIMIT_REQUEST_FIELD_SIZE` (8190 as distributed) that will be allowed in an HTTP request header.

The LimitRequestFieldsize directive allows the server administrator to reduce the limit on the allowed size of an HTTP request header field below the normal input buffer size compiled with the server. A server needs this value to be large enough to hold any one header field from a normal client request. The size of a normal request header field will vary greatly among different client implementations, often depending upon the extent to which a user has configured their browser to support detailed content negotiation.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks. Under normal conditions, the value should not be changed from the default.

LimitRequestLine directive

Syntax: LimitRequestLine *bytes*

Default: LimitRequestLine 8190

Context: server config

Status: core

Compatibility: LimitRequestLine is only available in Apache 1.3.2 and later.

This directive sets the number of *bytes* from 0 to the value of the compile-time constant `DEFAULT_LIMIT_REQUEST_LINE`

(8190 as distributed) that will be allowed on the HTTP request-line.

The `LimitRequestLine` directive allows the server administrator to reduce the limit on the allowed size of a client's HTTP request-line below the normal input buffer size compiled with the server. Since the request-line consists of the HTTP method, URI, and protocol version, the `LimitRequestLine` directive places a restriction on the length of a request-URI allowed for a request on the server. A server needs this value to be large enough to hold any of its resource names, including any information that might be passed in the query part of a GET request.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks. Under normal conditions, the value should not be changed from the default.

Listen directive

Syntax: `Listen [IP-address:]port`

Context: server config

Status: core

Compatibility: Listen is only available in Apache 1.1 and later.

The Listen directive instructs Apache to listen to more than one IP address or port; by default it responds to requests on all IP interfaces, but only on the port given by the `Port` directive.

Listen can be used instead of `BindAddress` and `Port`. It tells the server to accept incoming requests on the specified port or address-and-port combination. If the first format is used, with a port number only, the server listens to the given port on all interfaces, instead of the port given by the `Port` directive. If an IP address is given as well as a port, the server will listen on the given port and interface.

Note that you may still require a `Port` directive so that URLs that Apache generates that point to your server still work.

Multiple Listen directives may be used to specify a number of addresses and ports to listen to. The server will respond to requests from any of the listed addresses and ports.

For example, to make the server accept connections on both port 80 and port 8000, use:

```
Listen 80
Listen 8000
```

To make the server accept connections on two specified interfaces and port numbers, use

```
Listen 192.170.2.1:80
Listen 192.170.2.5:8000
```

See Also: [DNS Issues](#)

See Also: [Setting which addresses and ports Apache uses](#)

See Also: [Known Bugs](#)

ListenBacklog directive

Syntax: `ListenBacklog backlog`

Default: `ListenBacklog 511`

Context: server config

Status: Core

Compatibility: ListenBacklog is only available in Apache versions after 1.2.0.

The maximum length of the queue of pending connections. Generally no tuning is needed or desired, however on some systems it is desirable to increase this when under a TCP SYN flood attack. See the backlog parameter to the `listen(2)` system call.

This will often be limited to a smaller number by the operating system. This varies from OS to OS. Also note that many OSes do not use exactly what is specified as the backlog, but use a number based on (but normally larger than) what is set.

<Location> directive

Syntax: <Location *URL-path|URL*> ... </Location>

Context: server config, virtual host

Status: core

Compatibility: Location is only available in Apache 1.1 and later.

The <Location> directive provides for access control by URL. It is similar to the <Directory> directive, and starts a subsection which is terminated with a </Location> directive. <Location> sections are processed in the order they appear in the configuration file, after the <Directory> sections and .htaccess files are read, and after the <Files> sections.

Note that URLs do not have to line up with the filesystem at all, it should be emphasized that <Location> operates completely outside the filesystem.

For all origin (non-proxy) requests, the URL to be matched is of the form `/path/`, and you should not include any `http://servername` prefix. For proxy requests, the URL to be matched is of the form `scheme://servername/path`, and you must include the prefix.

The URL may use wildcards. In a wild-card string, `'?'` matches any single character, and `'*'` matches any sequences of characters.

Apache 1.2 and above: Extended regular expressions can also be used, with the addition of the `~` character. For example:

```
<Location ~ "/(extra|special)/data">
```

would match URLs that contained the substring `"/extra/data"` or `"/special/data"`. In Apache 1.3 and above, a new directive <LocationMatch> exists which behaves identical to the regex version of <Location>.

The Location functionality is especially useful when combined with the <SetHandler> directive. For example, to enable status requests, but allow them only from browsers at `foo.com`, you might use:

```
<Location /status>
SetHandler server-status
Order Deny,Allow
Deny from all
Allow from .foo.com
</Location>
```

Apache 1.3 and above note about / (slash): The slash character has special meaning depending on where in a URL it appears. People may be used to its behavior in the filesystem where multiple adjacent slashes are frequently collapsed to a single slash (*i.e.*, `/home///foo` is the same as `/home/foo`). In URL-space this is not necessarily true. The <LocationMatch> directive and the regex version of <Location> require you to explicitly specify multiple slashes if that is your intention. For example, <LocationMatch ^/abc> would match the request URL `/abc` but not the request URL `//abc`. The (non-regex) <Location> directive behaves similarly when used for proxy requests. But when (non-regex) <Location> is used for non-proxy requests it will implicitly match multiple slashes with a single slash. For example, if you specify <Location /abc/def> and the request is to `/abc//def` then it will match.

See also: [How Directory, Location and Files sections work](#) for an explanation of how these different sections are combined when a request is received

<LocationMatch>

Syntax: <LocationMatch *regex*> ... </LocationMatch>

Context: server config, virtual host

Status: core

Compatibility: LocationMatch is only available in Apache 1.3 and later.

The <LocationMatch> directive provides for access control by URL, in an identical manner to [<Location>](#). However, it takes a regular expression as an argument instead of a simple string. For example:

```
<LocationMatch "/(extra|special)/data">
```

would match URLs that contained the substring "/extra/data" or "/special/data".

See also: [How Directory, Location and Files sections work](#) for an explanation of how these different sections are combined when a request is received

LockFile directive

Syntax: LockFile *file-path*

Default: LockFile logs/accept.lock

Context: server config

Status: core

The LockFile directive sets the path to the lockfile used when Apache is compiled with either USE_FCNTL_SERIALIZED_ACCEPT or USE_FLOCK_SERIALIZED_ACCEPT. This directive should normally be left at its default value. The main reason for changing it is if the logs directory is NFS mounted, since **the lockfile must be stored on a local disk**. The PID of the main server process is automatically appended to the filename.

SECURITY: It is best to avoid putting this file in a world writable directory such as /var/tmp because someone could create a denial of service attack and prevent the server from starting by creating a lockfile with the same name as the one the server will try to create.

LogLevel directive

Syntax: LogLevel *level*

Default: LogLevel warn

Context: server config, virtual host

Status: core

Compatibility: LogLevel is only available in 1.3 or later.

LogLevel adjusts the verbosity of the messages recorded in the error logs (see [ErrorLog](#) directive). The following *levels* are available, in order of decreasing significance:

Level	Description
	Example
emerg	Emergencies - system is unusable. "Child cannot open lock file. Exiting"
alert	Action must be taken immediately. "getpwuid: couldn't determine user name from uid"
crit	Critical Conditions. "socket: Failed to get a socket, exiting child"
error	Error conditions.

	"Premature end of script headers"
warn	Warning conditions. "child process 1234 did not exit, sending another SIGHUP"
notice	Normal but significant condition. "httpd: caught SIGBUS, attempting to dump core in ..."
info	Informational. "Server seems busy, (you may need to increase StartServers, or Min/MaxSpareServers)..."
debug	Debug-level messages "Opening config file ..."

When a particular level is specified, messages from all other levels of higher significance will be reported as well. *E.g.*, when `LogLevel info` is specified, then messages with log levels of `notice` and `warn` will also be posted.

Using a level of at least `crit` is recommended.

MaxClients directive

Syntax: `MaxClients` *number*

Default: `MaxClients` 256

Context: server config

Status: core

The `MaxClients` directive sets the limit on the number of simultaneous requests that can be supported; not more than this number of child server processes will be created. To configure more than 256 clients, you must edit the `HARD_SERVER_LIMIT` entry in `httpd.h` and recompile.

Any connection attempts over the `MaxClients` limit will normally be queued, up to a number based on the [ListenBacklog](#) directive. Once a child process is freed at the end of a different request, the connection will then be serviced.

MaxKeepAliveRequests directive

Syntax: `MaxKeepAliveRequests` *number*

Default: `MaxKeepAliveRequests` 100

Context: server config

Status: core

Compatibility: Only available in Apache 1.2 and later.

The `MaxKeepAliveRequests` directive limits the number of requests allowed per connection when [KeepAlive](#) is on. If it is set to "0", unlimited requests will be allowed. We recommend that this setting be kept to a high value for maximum server performance. In Apache 1.1, this is controlled through an option to the `KeepAlive` directive.

MaxRequestsPerChild directive

Syntax: `MaxRequestsPerChild` *number*

Default: `MaxRequestsPerChild` 0

Context: server config

Status: core

The `MaxRequestsPerChild` directive sets the limit on the number of requests that an individual child server process will handle. After `MaxRequestsPerChild` requests, the child process will die. If `MaxRequestsPerChild` is 0, then the process will never expire.

Setting `MaxRequestsPerChild` to a non-zero limit has two beneficial effects:

- it limits the amount of memory that process can consume by (accidental) memory leakage;
- by giving processes a finite lifetime, it helps reduce the number of processes when the server load reduces.

However, on Win32, It is recommended that this be set to 0. If it is set to a non-zero value, when the request count is reached, the child process exits, and is respawned, at which time it re-reads the configuration files. This can lead to unexpected behavior if you have modified a configuration file, but are not expecting the changes to be applied yet. See also [ThreadsPerChild](#).

NOTE: For *KeepAlive* requests, only the first request is counted towards this limit. In effect, it changes the behavior to limit the number of *connections* per child.

MaxSpareServers directive

Syntax: `MaxSpareServers` *number*

Default: `MaxSpareServers` 10

Context: server config

Status: core

The `MaxSpareServers` directive sets the desired maximum number of *idle* child server processes. An idle process is one which is not handling a request. If there are more than `MaxSpareServers` idle, then the parent process will kill off the excess processes.

Tuning of this parameter should only be necessary on very busy sites. Setting this parameter to a large number is almost always a bad idea.

This directive has no effect when used with the Apache Web server on a Microsoft Windows platform.

See also [MinSpareServers](#) and [StartServers](#).

MinSpareServers directive

Syntax: `MinSpareServers` *number*

Default: `MinSpareServers` 5

Context: server config

Status: core

The `MinSpareServers` directive sets the desired minimum number of *idle* child server processes. An idle process is one which is not handling a request. If there are fewer than `MinSpareServers` idle, then the parent process creates new children at a maximum rate of 1 per second.

Tuning of this parameter should only be necessary on very busy sites. Setting this parameter to a large number is almost always a bad idea.

This directive has no effect on Microsoft Windows.

See also [MaxSpareServers](#) and [StartServers](#).

NameVirtualHost directive

Syntax: `NameVirtualHost` *addr[:port]*

Context: server config

Status: core

Compatibility: NameVirtualHost is only available in Apache 1.3 and later

The NameVirtualHost directive is a required directive if you want to configure [name-based virtual hosts](#).

Although *addr* can be hostname it is recommended that you always use an IP address or wildcard, *e.g.*

```
NameVirtualHost 111.22.33.44
```

With the NameVirtualHost directive you specify the IP address on which the server will receive requests for the name-based virtual hosts. This will usually be the address to which your name-based virtual host names resolve. In cases where a firewall or other proxy receives the requests and forwards them on a different IP address to the server, you must specify the IP address of the physical interface on the machine which will be servicing the requests. If you have multiple name-based hosts on multiple addresses, repeat the directive for each address.

Note: the "main server" and any `_default_` servers will **never** be served for a request to a NameVirtualHost IP Address (unless for some reason you specify NameVirtualHost but then don't define any VirtualHosts for that address).

Optionally you can specify a port number on which the name-based virtual hosts should be used, *e.g.*

```
NameVirtualHost 111.22.33.44:8080
```

In Apache 1.3.13 and greater you can specify a `*` for the *addr*. This creates a wildcard NameVirtualHost which will match connections to any address that isn't configured with a more specific NameVirtualHost directive or [<VirtualHost>](#) section. This is useful if you want only name-based virtual hosts and you don't want to hard-code the server's IP address into the configuration file.

See also: [Apache Virtual Host documentation](#)

Options directive

Syntax: Options [+|-]*option* [[+|-]*option*] ...

Context: server config, virtual host, directory, .htaccess

Override: Options

Status: core

The Options directive controls which server features are available in a particular directory.

option can be set to `None`, in which case none of the extra features are enabled, or one or more of the following:

All

All options except for MultiViews. This is the default setting.

ExecCGI

Execution of CGI scripts is permitted.

FollowSymLinks

The server will follow symbolic links in this directory.

Note: even though the server follows the symlink it does *not* change the pathname used to match against `<Directory>` sections.

Note: this option gets ignored if set inside a `<Location>` section.

Includes

Server-side includes are permitted.

IncludesNOEXEC

Server-side includes are permitted, but the `#exec` command and `#exec` CGI are disabled. It is still possible to `#include` virtual CGI scripts from ScriptAlias'd directories.

Indexes

If a URL which maps to a directory is requested, and there is no `DirectoryIndex` (*e.g.*, `index.html`) in that directory, then the server will return a formatted listing of the directory.

MultiViews

[Content negotiated](#) MultiViews are allowed.

SymLinksIfOwnerMatch

The server will only follow symbolic links for which the target file or directory is owned by the same user id as the link.

Note: this option gets ignored if set inside a <Location> section.

Normally, if multiple Options could apply to a directory, then the most specific one is taken complete; the options are not merged. However if *all* the options on the Options directive are preceded by a + or - symbol, the options are merged. Any options preceded by a + are added to the options currently in force, and any options preceded by a - are removed from the options currently in force.

For example, without any + and - symbols:

```
<Directory /web/docs>
Options Indexes FollowSymLinks
</Directory>
<Directory /web/docs/spec>
Options Includes
</Directory>
```

then only Includes will be set for the /web/docs/spec directory. However if the second Options directive uses the + and - symbols:

```
<Directory /web/docs>
Options Indexes FollowSymLinks
</Directory>
<Directory /web/docs/spec>
Options +Includes -Indexes
</Directory>
```

then the options FollowSymLinks and Includes are set for the /web/docs/spec directory.

Note: Using -IncludesNOEXEC or -Includes disables server-side includes completely regardless of the previous setting.

The default in the absence of any other settings is All.

PidFile directive

Syntax: PidFile *file-path*

Default: PidFile logs/httpd.pid

Context: server config

Status: core

The PidFile directive sets the file to which the server records the process id of the daemon. If the filename does not begin with a slash (/) then it is assumed to be relative to the [ServerRoot](#). The PidFile is only used in [standalone](#) mode.

It is often useful to be able to send the server a signal, so that it closes and then reopens its [ErrorLog](#) and TransferLog, and re-reads its configuration files. This is done by sending a SIGHUP (kill -1) signal to the process id listed in the PidFile.

The PidFile is subject to the same warnings about log file placement and [security](#).

Port directive

Syntax: Port *number*

Default: Port 80

Context: server config

Status: core

Number is a number from 0 to 65535; some port numbers (especially below 1024) are reserved for particular protocols. See

`/etc/services` for a list of some defined ports; the standard port for the http protocol is 80.

The `Port` directive has two behaviors, the first of which is necessary for NCSA backwards compatibility (and which is confusing in the context of Apache).

- In the absence of any [Listen](#) or [BindAddress](#) directives specifying a port number, a `Port` directive given in the "main server" (*i.e.*, outside any `<VirtualHost>` section) sets the network port on which the server listens. If there are any `Listen` or `BindAddress` directives specifying `:number` then `Port` has no effect on what address the server listens at.
- The `Port` directive sets the `SERVER_PORT` environment variable (for [CGI](#) and [SSI](#)), and is used when the server must generate a URL that refers to itself (for example when creating an external redirect to itself). This behavior is modified by [UseCanonicalName](#).

The primary behavior of `Port` should be considered to be similar to that of the [ServerName](#) directive. The `ServerName` and `Port` together specify what you consider to be the *canonical* address of the server. (See also [UseCanonicalName](#).)

Port 80 is one of Unix's special ports. All ports numbered below 1024 are reserved for system use, *i.e.*, regular (non-root) users cannot make use of them; instead they can only use higher port numbers. To use port 80, you must start the server from the root account. After binding to the port and before accepting requests, Apache will change to a low privileged user as set by the [User](#) directive.

If you cannot use port 80, choose any other unused port. Non-root users will have to choose a port number higher than 1023, such as 8000.

SECURITY: if you do start the server as root, be sure not to set [User](#) to root. If you run the server as root whilst handling connections, your site may be open to a major security attack.

Require directive

Syntax: `Require entity-name [entity-name] ...`

Context: directory, `.htaccess`

Override: `AuthConfig`

Status: core

This directive selects which authenticated users can access a directory. The allowed syntaxes are:

- `Require user userid [userid] ...`
Only the named users can access the directory.
- `Require group group-name [group-name] ...`
Only users in the named groups can access the directory.
- `Require valid-user`

All valid users can access the directory.

`Require` must be accompanied by [AuthName](#) and [AuthType](#) directives, and directives such as [AuthUserFile](#) and [AuthGroupFile](#) (to define users and groups) in order to work correctly. Example:

```
AuthType Basic
AuthName "Restricted Directory"
AuthUserFile /web/users
AuthGroupFile /web/groups
Require group admin
```

Access controls which are applied in this way are effective for **all** methods. **This is what is normally desired.** If you wish to apply access controls only to specific methods, while leaving other methods unprotected, then place the `Require` statement into a `<Limit>` section

See also [Satisfy](#) and [mod_access](#).

ResourceConfig directive

Syntax: ResourceConfig *file-path|directory-path*

Default: ResourceConfig conf/srm.conf

Context: server config, virtual host

Status: core

Compatibility: The ability to specify a directory, rather than a file name, is only available in Apache 1.3.13 and later.

The server will read this file for more directives after reading the httpd.conf file. *File-path* is relative to the [ServerRoot](#). This feature can be disabled using:

```
ResourceConfig /dev/null
```

Or, on Win32 servers,

```
ResourceConfig nul
```

Historically, this file contained most directives except for server configuration directives and [<Directory>](#) sections; in fact it can now contain any server directive allowed in the *server config* context. However, since Apache version 1.3.4, the default srm.conf file which ships with Apache contains only comments, and all directives are placed in the main server configuration file, httpd.conf.

If ResourceConfig points to a directory, rather than a file, Apache will read all files in that directory, and any subdirectory, and parse those as configuration files. Note that any file in the specified directory will be loaded as a configuration file, so make sure that you don't have any stray files in this directory by mistake, such as temporary files created by your editor, for example.

See also [AccessConfig](#).

RLimitCPU directive

Syntax: RLimitCPU *number|max* [*number|max*]

Default: Unset; uses operating system defaults

Context: server config, virtual host

Status: core

Compatibility: RLimitCPU is only available in Apache 1.2 and later

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes and the second parameter sets the maximum resource limit. Either parameter can be a number, or max to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as root, or in the initial startup phase.

This applies to processes forked off from Apache children servicing requests, not the Apache children themselves. This includes CGI scripts and SSI exec commands, but not any processes forked off from the Apache parent such as piped logs.

CPU resource limits are expressed in seconds per process.

See also [RLimitMEM](#) or [RLimitNPROC](#).

RLimitMEM directive

Syntax: RLimitMEM *number|max* [*number|max*]

Default: Unset; uses operating system defaults

Context: server config, virtual host

Status: core

Compatibility: RLimitMEM is only available in Apache 1.2 and later

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes and the second parameter sets the maximum resource limit. Either parameter can be a number, or `max` to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as root, or in the initial startup phase.

This applies to processes forked off from Apache children servicing requests, not the Apache children themselves. This includes CGI scripts and SSI exec commands, but not any processes forked off from the Apache parent such as piped logs.

Memory resource limits are expressed in bytes per process.

See also [RLimitCPU](#) or [RLimitNPROC](#).

RLimitNPROC directive

Syntax: RLimitNPROC *number*|max [*number*|max]

Default: *Unset; uses operating system defaults*

Context: server config, virtual host

Status: core

Compatibility: RLimitNPROC is only available in Apache 1.2 and later

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes and the second parameter sets the maximum resource limit. Either parameter can be a number, or `max` to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as root, or in the initial startup phase.

This applies to processes forked off from Apache children servicing requests, not the Apache children themselves. This includes CGI scripts and SSI exec commands, but not any processes forked off from the Apache parent such as piped logs.

Process limits control the number of processes per user.

Note: If CGI processes are **not** running under userids other than the web server userid, this directive will limit the number of processes that the server itself can create. Evidence of this situation will be indicated by *cannot fork* messages in the `error_log`.

See also [RLimitMEM](#) or [RLimitCPU](#).

Satisfy directive

Syntax: Satisfy any|all

Default: Satisfy all

Context: directory, `.htaccess`

Status: core

Compatibility: Satisfy is only available in Apache 1.2 and later

Access policy if both `Allow` and `Require` used. The parameter can be either *'all'* or *'any'*. This directive is only useful if access to a particular area is being restricted by both username/password *and* client host address. In this case the default behavior ("all") is to require that the client passes the address access restriction *and* enters a valid username and password. With the "any" option the client will be granted access if they either pass the host restriction or enter a valid username and password. This can be used to password restrict an area, but to let clients from particular addresses in without prompting for a password.

See also [Require](#) and [Allow](#).

ScoreBoardFile directive

Syntax: ScoreBoardFile *file-path*

Default: ScoreBoardFile logs/apache_status

Context: server config

Status: core

The ScoreBoardFile directive is required on some architectures to place a file that the server will use to communicate between its children and the parent. The easiest way to find out if your architecture requires a scoreboard file is to run Apache and see if it creates the file named by the directive. If your architecture requires it then you must ensure that this file is not used at the same time by more than one invocation of Apache.

If you have to use a ScoreBoardFile then you may see improved speed by placing it on a RAM disk. But be careful that you heed the same warnings about log file placement and [security](#).

Apache 1.2 and above:

Linux 1.x users might be able to add `-DHAVE_SHMGET -DUSE_SHMGET_SCOREBOARD` to the `EXTRA_CFLAGS` in your Configuration. This might work with some 1.x installations, but won't work with all of them. (Prior to 1.3b4, `HAVE_SHMGET` would have sufficed.)

SVR4 users should consider adding `-DHAVE_SHMGET -DUSE_SHMGET_SCOREBOARD` to the `EXTRA_CFLAGS` in your Configuration. This is believed to work, but we were unable to test it in time for 1.2 release. (Prior to 1.3b4, `HAVE_SHMGET` would have sufficed.)

See Also: [Stopping and Restarting Apache](#)

ScriptInterpreterSource directive

Syntax: ScriptInterpreterSource registry|script

Default: ScriptInterpreterSource script

Context: directory, .htaccess

Status: core (Windows only)

This directive is used to control how Apache 1.3.5 and later finds the interpreter used to run CGI scripts. The default technique is to use the interpreter pointed to by the `#!` line in the script. Setting `ScriptInterpreterSource` registry will cause the Windows Registry to be searched using the script file extension (e.g., `.pl`) as a search key.

SendBufferSize directive

Syntax: SendBufferSize *bytes*

Context: server config

Status: core

The server will set the TCP buffer size to the number of bytes specified. Very useful to increase past standard OS defaults on high speed high latency (*i.e.*, 100ms or so, such as transcontinental fast pipes)

ServerAdmin directive

Syntax: `ServerAdmin email-address`

Context: server config, virtual host

Status: core

The `ServerAdmin` sets the e-mail address that the server includes in any error messages it returns to the client.

It may be worth setting up a dedicated address for this, *e.g.*

```
ServerAdmin www-admin@foo.bar.com
```

as users do not always mention that they are talking about the server!

ServerAlias directive

Syntax: `ServerAlias hostname [hostname] ...`

Context: virtual host

Status: core

Compatibility: `ServerAlias` is only available in Apache 1.1 and later.

The `ServerAlias` directive sets the alternate names for a host, for use with [name-based virtual hosts](#).

See also: [Apache Virtual Host documentation](#)

ServerName directive

Syntax: `ServerName fully-qualified-domain-name`

Context: server config, virtual host

Status: core

The `ServerName` directive sets the hostname of the server; this is used when creating redirection URLs. If it is not specified, then the server attempts to deduce it from its own IP address; however this may not work reliably, or may not return the preferred hostname. For example:

```
ServerName www.example.com
```

would be used if the canonical (main) name of the actual machine were `simple.example.com`.

If you are using [name-based virtual hosts](#), the `ServerName` inside a `<VirtualHost>` section specifies what hostname must appear in the request's `Host :` header to match this virtual host.

See Also:

[DNS Issues](#)

[Apache virtual host documentation](#)

[UseCanonicalName](#)

[NameVirtualHost](#)

[ServerAlias](#)

ServerPath directive

Syntax: `ServerPath` *directory-path*

Context: virtual host

Status: core

Compatibility: ServerPath is only available in Apache 1.1 and later.

The ServerPath directive sets the legacy URL pathname for a host, for use with [name-based virtual hosts](#).

See also: [Apache Virtual Host documentation](#)

ServerRoot directive

Syntax: `ServerRoot` *directory-path*

Default: `ServerRoot /usr/local/apache`

Context: server config

Status: core

The ServerRoot directive sets the directory in which the server lives. Typically it will contain the subdirectories `conf/` and `logs/`. Relative paths for other configuration files are taken as relative to this directory.

See also [the `-d` option to `httpd`](#).

See also [the security tips](#) for information on how to properly set permissions on the ServerRoot.

ServerSignature directive

Syntax: `ServerSignature` On|Off|EMail

Default: `ServerSignature Off`

Context: server config, virtual host, directory, `.htaccess`

Status: core

Compatibility: ServerSignature is only available in Apache 1.3 and later.

The ServerSignature directive allows the configuration of a trailing footer line under server-generated documents (error messages, `mod_proxy` ftp directory listings, `mod_info` output, ...). The reason why you would want to enable such a footer line is that in a chain of proxies, the user often has no possibility to tell which of the chained servers actually produced a returned error message. The Off setting, which is the default, suppresses the error line (and is therefore compatible with the behavior of Apache-1.2 and below). The On setting simply adds a line with the server version number and [ServerName](#) of the serving virtual host, and the EMail setting additionally creates a "mailto:" reference to the [ServerAdmin](#) of the referenced document.

ServerTokens directive

Syntax: `ServerTokens` Minimal|ProductOnly|OS|Full

Default: `ServerTokens Full`

Context: server config

Status: core

Compatibility: ServerTokens is only available in Apache 1.3 and later; the `ProductOnly` keyword is only available in versions later than 1.3.12

This directive controls whether Server response header field which is sent back to clients includes a description of the generic

OS-type of the server as well as information about compiled-in modules.

`ServerTokens Prod[uctOnly]`

Server sends (e.g.): Server: Apache

`ServerTokens Min[imal]`

Server sends (e.g.): Server: Apache/1.3.0

`ServerTokens OS`

Server sends (e.g.): Server: Apache/1.3.0 (Unix)

`ServerTokens Full` (or not specified)

Server sends (e.g.): Server: Apache/1.3.0 (Unix) PHP/3.0 MyMod/1.2

This setting applies to the entire server, and cannot be enabled or disabled on a virtualhost-by-virtualhost basis.

ServerType directive

Syntax: `ServerType type`

Default: `ServerType standalone`

Context: server config

Status: core

The `ServerType` directive sets how the server is executed by the system. *Type* is one of `inetd`

The server will be run from the system process `inetd`; the command to start the server is added to `/etc/inetd.conf`

`standalone`

The server will run as a daemon process; the command to start the server is added to the system startup scripts. (`/etc/rc.local` or `/etc/rc3.d/...`)

`inetd` is the lesser used of the two options. For each http connection received, a new copy of the server is started from scratch; after the connection is complete, this program exits. There is a high price to pay per connection, but for security reasons, some admins prefer this option. **Inetd mode is no longer recommended and does not always work properly. Avoid it if at all possible.**

`standalone` is the most common setting for `ServerType` since it is far more efficient. The server is started once, and services all subsequent connections. If you intend running Apache to serve a busy site, `standalone` will probably be your only option.

StartServers directive

Syntax: `StartServers number`

Default: `StartServers 5`

Context: server config

Status: core

The `StartServers` directive sets the number of child server processes created on startup. As the number of processes is dynamically controlled depending on the load, there is usually little reason to adjust this parameter.

When running under Microsoft Windows, this directive has no effect. There is always one child which handles all requests. Within the child requests are handled by separate threads. The [ThreadsPerChild](#) directive controls the maximum number of child threads handling requests, which will have a similar effect to the setting of `StartServers` on Unix.

See also [MinSpareServers](#) and [MaxSpareServers](#).

ThreadsPerChild

Syntax: ThreadsPerChild *number*

Default: ThreadsPerChild 50

Context: server config

Status: core (Windows, NetWare)

Compatibility: Available only with Apache 1.3 and later with Windows

This directive tells the server how many threads it should use. This is the maximum number of connections the server can handle at once; be sure and set this number high enough for your site if you get a lot of hits.

This directive has no effect on Unix systems. Unix users should look at [StartServers](#) and [MaxRequestsPerChild](#).

ThreadStackSize

Syntax: ThreadStackSize *number*

Default: ThreadStackSize 65536

Context: server config

Status: core (NetWare)

Compatibility: Available only with Apache 1.3 and later with NetWare

This directive tells the server what stack size to use for each of the running threads. If you ever get a stack overflow you will need to bump this number to a higher setting.

This directive has no effect on other systems.

TimeOut directive

Syntax: TimeOut *number*

Default: TimeOut 300

Context: server config

Status: core

The TimeOut directive currently defines the amount of time Apache will wait for three things:

1. The total amount of time it takes to receive a GET request.
2. The amount of time between receipt of TCP packets on a POST or PUT request.
3. The amount of time between ACKs on transmissions of TCP packets in responses.

We plan on making these separately configurable at some point down the road. The timer used to default to 1200 before 1.2, but has been lowered to 300 which is still far more than necessary in most situations. It is not set any lower by default because there may still be odd places in the code where the timer is not reset when a packet is sent.

UseCanonicalName directive

Syntax: UseCanonicalName on|off|dns

Default: UseCanonicalName on

Context: server config, virtual host, directory

Override: Options

Compatibility: UseCanonicalName is only available in Apache 1.3 and later

In many situations Apache has to construct a *self-referential* URL. That is, a URL which refers back to the same server. With

`UseCanonicalName on` (and in all versions prior to 1.3) Apache will use the [ServerName](#) and [Port](#) directives to construct a canonical name for the server. This name is used in all self-referential URLs, and for the values of `SERVER_NAME` and `SERVER_PORT` in CGIs.

With `UseCanonicalName off` Apache will form self-referential URLs using the hostname and port supplied by the client if any are supplied (otherwise it will use the canonical name). These values are the same that are used to implement [name based virtual hosts](#), and are available with the same clients. The CGI variables `SERVER_NAME` and `SERVER_PORT` will be constructed from the client supplied values as well.

An example where this may be useful is on an intranet server where you have users connecting to the machine using short names such as `www`. You'll notice that if the users type a shortname, and a URL which is a directory, such as `http://www/splat`, *without the trailing slash* then Apache will redirect them to `http://www.domain.com/splat/`. If you have authentication enabled, this will cause the user to have to reauthenticate twice (once for `www` and once again for `www.domain.com`). But if `UseCanonicalName` is set off, then Apache will redirect to `http://www/splat/`.

There is a third option, `UseCanonicalName DNS`, which is intended for use with mass IP-based virtual hosting to support ancient clients that do not provide a `Host:` header. With this option Apache does a reverse DNS lookup on the server IP address that the client connected to in order to work out self-referential URLs.

Warning: if CGIs make assumptions about the values of `SERVER_NAME` they may be broken by this option. The client is essentially free to give whatever value they want as a hostname. But if the CGI is only using `SERVER_NAME` to construct self-referential URLs then it should be just fine.

See also: [ServerName](#), [Port](#)

User directive

Syntax: `User unix-userid`

Default: `User #-1`

Context: server config, virtual host

Status: core

The User directive sets the userid as which the server will answer requests. In order to use this directive, the standalone server must be run initially as root. *Unix-userid* is one of:

A username

Refers to the given user by name.

followed by a user number.

Refers to a user by their number.

The user should have no privileges which result in it being able to access files which are not intended to be visible to the outside world, and similarly, the user should not be able to execute code which is not meant for httpd requests. It is recommended that you set up a new user and group specifically for running the server. Some admins use user `nobody`, but this is not always possible or desirable. For example `mod_proxy`'s cache, when enabled, must be accessible to this user (see the [CacheRoot directive](#)).

Notes: If you start the server as a non-root user, it will fail to change to the lesser privileged user, and will instead continue to run as that original user. If you do start the server as root, then it is normal for the parent process to remain running as root.

Special note: Use of this directive in `<VirtualHost>` requires a properly configured [suEXEC wrapper](#). When used inside a `<VirtualHost>` in this manner, only the user that CGIs are run as is affected. Non-CGI requests are still processed with the user specified in the main User directive.

SECURITY: Don't set User (or [Group](#)) to `root` unless you know exactly what you are doing, and what the dangers are.

<VirtualHost> directive

Syntax: <VirtualHost *addr[:port] [addr[:port]] ...> ... </VirtualHost>*

Context: server config

Status: Core.

Compatibility: Non-IP address-based Virtual Hosting only available in Apache 1.1 and later.

Compatibility: Multiple address support only available in Apache 1.2 and later.

<VirtualHost> and </VirtualHost> are used to enclose a group of directives which will apply only to a particular virtual host. Any directive which is allowed in a virtual host context may be used. When the server receives a request for a document on a particular virtual host, it uses the configuration directives enclosed in the <VirtualHost> section. *Addr* can be

- The IP address of the virtual host
- A fully qualified domain name for the IP address of the virtual host.

Example:

```
<VirtualHost 10.1.2.3>
ServerAdmin webmaster@host.foo.com
DocumentRoot /www/docs/host.foo.com
ServerName host.foo.com
ErrorLog logs/host.foo.com-error_log
TransferLog logs/host.foo.com-access_log
</VirtualHost>
```

Each VirtualHost must correspond to a different IP address, different port number or a different host name for the server, in the former case the server machine must be configured to accept IP packets for multiple addresses. (If the machine does not have multiple network interfaces, then this can be accomplished with the `ifconfig alias` command (if your OS supports it), or with kernel patches like [VIF](#) (for SunOS(TM) 4.1.x)).

You can specify more than one IP address. This is useful if a machine responds to the same name on two different interfaces. For example, if you have a VirtualHost that is available to hosts on an internal (intranet) as well as external (internet) network. Example:

```
<VirtualHost 192.168.1.2 204.255.176.199>
DocumentRoot /www/docs/host.foo.com
ServerName host.foo.com
ServerAlias host
</VirtualHost>
```

The special name `_default_` can be specified in which case this virtual host will match any IP address that is not explicitly listed in another virtual host. In the absence of any `_default_` virtual host the "main" server config, consisting of all those definitions outside any VirtualHost section, is used when no match occurs.

You can specify a `:port` to change the port that is matched. If unspecified then it defaults to the same port as the most recent [Port](#) statement of the main server. You may also specify `:*` to match all ports on that address. (This is recommended when used with `_default_`.)

SECURITY: See the [security tips](#) document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

NOTE: The use of <VirtualHost> does **not** affect what addresses Apache listens on. You may need to ensure that Apache is listening on the correct addresses using either [BindAddress](#) or [Listen](#).

See also: [Apache Virtual Host documentation](#)

See also: [Warnings about DNS and Apache](#)

See also: [Setting which addresses and ports Apache uses](#)

See also: [How Directory, Location and Files sections work](#) for an explanation of how these different sections are combined when a request is received

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_access

This module provides access control based on client hostname, IP address, or other characteristics of the client request.

Status: Base

Source File: mod_access.c

Module Identifier: access_module

Summary

The directives provided by mod_access are used in `<Directory>`, `<Files>`, and `<Location>` sections as well as `.htaccess` files to control access to particular parts of the server. Access can be controlled based on the client hostname, IP address, or other characteristics of the client request, as captured in [environment variables](#). The `Allow` and `Deny` directives are used to specify which clients are or are not allowed access to the server, while the `Order` directive sets the default access state, and configures how the `Allow` and `Deny` directives interact with each other.

Both host-based access restrictions and password-based authentication may be implemented simultaneously. In that case, the [Satisfy](#) directive is used to determine how the two sets of restrictions interact.

In general, access restriction directives apply to all access methods (GET, PUT, POST, etc). This is the desired behavior in most cases. However, it is possible to restrict some methods, while leaving other methods unrestricted, by enclosing the directives in a `<Limit>` section.

Directives

- [Allow](#)
- [Deny](#)
- [Order](#)

See also [Satisfy](#) and [Require](#).

Allow directive

Syntax: Allow from all[*host|env=env-variable* [*host|env=env-variable*] ...

Context: directory, .htaccess

Override: Limit

Status: Base

Module: mod_access

The `Allow` directive affects which hosts can access an area of the server. Access can be controlled by hostname, IP Address, IP Address range, or by other characteristics of the client request captured in environment variables.

The first argument to this directive is always `from`. The subsequent arguments can take three different forms. If `Allow from`

all is specified, then all hosts are allowed access, subject to the configuration of the Deny and Order directives as discussed below. To allow only particular hosts or groups of hosts to access the server, the *host* can be specified in any of the following formats:

A (partial) domain-name

Example: Allow from apache.org

Hosts whose names match, or end in, this string are allowed access. Only complete components are matched, so the above example will match foo.apache.org but it will not match fooapache.org. This configuration will cause the server to perform a reverse DNS lookup on the client IP address, regardless of the setting of the [HostNameLookups](#) directive.

A full IP address

Example: Allow from 10.1.2.3

An IP address of a host allowed access

A partial IP address

Example: Allow from 10.1

The first 1 to 3 bytes of an IP address, for subnet restriction.

A network/netmask pair

Example: Allow from 10.1.0.0/255.255.0.0

A network a.b.c.d, and a netmask w.x.y.z. For more fine-grained subnet restriction. (Apache 1.3 and later)

A network/nnn CIDR specification

Example: Allow from 10.1.0.0/16

Similar to the previous case, except the netmask consists of nnn high-order 1 bits. (Apache 1.3 and later)

Note that the last three examples above match exactly the same set of hosts.

The third format of the arguments to the Allow directive allows access to the server to be controlled based on the existence of an [environment variable](#). When Allow from env=env-variable is specified, then the request is allowed access if the environment variable env-variable exists. The server provides the ability to set environment variables in a flexible way based on characteristics of the client request using the directives provided by [mod_setenvif](#). Therefore, this directive can be used to allow access based on such factors as the clients User-Agent (browser type), Referer, or other HTTP request header fields.

Example:

```
SetEnvIf User-Agent ^KnockKnock/2.0 let_me_in
<Directory /docroot>
    Order Deny,Allow
    Deny from all
    Allow from env=let_me_in
</Directory>
```

In this case, browsers with a user-agent string beginning with KnockKnock/2.0 will be allowed access, and all others will be denied.

See also [Deny](#), [Order](#) and [SetEnvIf](#).

Deny directive

Syntax: Deny from all|host|env=env-variable [host|env=env-variable] ...

Context: directory, .htaccess

Override: Limit

Status: Base

Module: mod_access

This directive allows access to the server to be restricted based on hostname, IP address, or environment variables. The arguments for the Deny directive are identical to the arguments for the [Allow](#) directive.

Order directive

Syntax: Order *ordering*

Default: Order Deny,Allow

Context: directory, .htaccess

Override: Limit

Status: Base

Module: mod_access

The Order directive controls the default access state and the order in which [Allow](#) and [Deny](#) directives are evaluated. *Ordering* is one of

Deny,Allow

The Deny directives are evaluated before the Allow directives. Access is allowed by default. Any client which does not match a Deny directive or does match an Allow directive will be allowed access to the server.

Allow,Deny

The Allow directives are evaluated before the Deny directives. Access is denied by default. Any client which does not match an Allow directive or does match a Deny directive will be denied access to the server.

Mutual-failure

Only those hosts which appear on the Allow list and do not appear on the Deny list are granted access. This ordering has the same effect as Order Allow,Deny and is deprecated in favor of that configuration.

Keywords may only be separated by a comma; no whitespace is allowed between them. Note that in all cases every Allow and Deny statement is evaluated.

In the following example, all hosts in the apache.org domain are allowed access; all other hosts are denied access.

```
Order Deny,Allow
Deny from all
Allow from apache.org
```

In the next example, all hosts in the apache.org domain are allowed access, except for the hosts which are in the foo.apache.org subdomain, who are denied access. All hosts not in the apache.org domain are denied access because the default state is to deny access to the server.

```
Order Allow,Deny
Allow from apache.org
Deny from foo.apache.org
```

On the other hand, if the Order in the last example is changed to Deny,Allow, all hosts will be allowed access. This happens because, regardless of the actual ordering of the directives in the configuration file, the Allow from apache.org will be evaluated last and will override the Deny from foo.apache.org. All hosts not in the apache.org domain will also be allowed access because the default state will change to *allow*.

The presence of an Order directive can affect access to a part of the server even in the absence of accompanying Allow and Deny directives because of its effect on the default access state. For example,

```
<Directory /www>
  Order Allow,Deny
</Directory>
```

will deny all access to the /www directory because the default access state will be set to *deny*.

The Order directive controls the order of access directive processing only within each phase of the server's configuration processing. This implies, for example, that an Allow or Deny directive occurring in a <Location> section will always be evaluated after an Allow or Deny directive occurring in a <Directory> section or .htaccess file, regardless of the setting of the Order directive. For details on the merging of configuration sections, see the documentation on [How Directory, Location and Files](#)

[sections work.](#)

See also: [Deny](#) and [Allow](#).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_actions

This module provides for executing CGI scripts based on media type or request method.

Status: Base

Source File: mod_actions.c

Module Identifier: actions_module

Compatibility: Available in Apache 1.1 and later.

Summary

This module has two directives. The Action directive lets you run CGI scripts whenever a file of a certain type is requested. The Script directive lets you run CGI scripts whenever a particular method is used in a request. This makes it much easier to execute scripts that process files.

Directives

- [Action](#)
- [Script](#)

Action directive

Syntax: Action *action-type* *cgi-script*

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_actions

Compatibility: Action is only available in Apache 1.1 and later

This directive adds an action, which will activate *cgi-script* when *action-type* is triggered by the request. The *action-type* can be either a [handler](#) or a MIME content type. It sends the URL and file path of the requested document using the standard CGI PATH_INFO and PATH_TRANSLATED environment variables.

Examples:

```
# Requests for files of a particular type:
Action image/gif /cgi-bin/images.cgi

# Files of a particular file extension
AddHandler my-file-type .xyz
Action my-file-type /cgi-bin/program.cgi
```

In the first example, requests for files with a MIME content type of image/gif will instead be handled by the specified cgi script

Module `mod_actions`

`/cgi-bin/images.cgi`.

In the second example, requests for files with a file extension of `.xyz` are handled instead by the specified cgi script `/cgi-bin/program.cgi`.

See also: [AddHandler](#)

Script directive

Syntax: `Script method cgi-script`

Context: server config, virtual host, directory

Status: Base

Module: `mod_actions`

Compatibility: Script is only available in Apache 1.1 and later; arbitrary method use is only available with 1.3.10 and later

This directive adds an action, which will activate *cgi-script* when a file is requested using the method of *method*. It sends the URL and file path of the requested document using the standard CGI `PATH_INFO` and `PATH_TRANSLATED` environment variables.

Prior to Apache 1.3.10, *method* can only be one of GET, POST, PUT, or DELETE. As of 1.3.10, any arbitrary method name may be used. **Method names are case-sensitive**, so `Script PUT` and `Script put` have two entirely different effects.

Note that the `Script` command defines default actions only. If a CGI script is called, or some other resource that is capable of handling the requested method internally, it will do so. Also note that `Script` with a method of GET will only be called if there are query arguments present (e.g., `foo.html?hi`). Otherwise, the request will proceed normally.

Examples:

```
# For <ISINDEX>-style searching
Script GET /cgi-bin/search
# A CGI PUT handler
Script PUT /~bob/put.cgi
```

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_alias

This module provides for mapping different parts of the host filesystem in the document tree, and for URL redirection.

Status: Base

Source File: mod_alias.c

Module Identifier: alias_module

Summary

The directives contained in this module allow for manipulation and control of URLs as requests arrive at the server. The `Alias` and `ScriptAlias` directives are used to map between URLs and filesystem paths. This allows for content which is not directly under the `DocumentRoot` to be served as part of the web document tree. The `ScriptAlias` directive has the additional effect of marking the target directory as containing only CGI scripts.

The `Redirect` directives are used to instruct clients to make a new request with a different URL. They are often used when a resource has moved to a new location.

A more powerful and flexible set of directives for manipulating URLs is contained in the [mod_rewrite](#) module.

Directives

- [Alias](#)
- [AliasMatch](#)
- [Redirect](#)
- [RedirectMatch](#)
- [RedirectTemp](#)
- [RedirectPermanent](#)
- [ScriptAlias](#)
- [ScriptAliasMatch](#)

Alias directive

Syntax: `Alias URL-path file-path|directory-path`

Context: server config, virtual host

Status: Base

Module: mod_alias

The `Alias` directive allows documents to be stored in the local filesystem other than under the `DocumentRoot`. URLs with a (%-decoded) path beginning with `url-path` will be mapped to local files beginning with `directory-filename`.

Example:

```
Alias /image /ftp/pub/image
```

A request for `http://myserver/image/foo.gif` would cause the server to return the file `/ftp/pub/image/foo.gif`.

Note that if you include a trailing `/` on the *url-path* then the server will require a trailing `/` in order to expand the alias. That is, if you use `Alias /icons/ /usr/local/apache/icons/` then the url `/icons` will not be aliased.

Note that you may need to specify additional [<Directory>](#) sections which cover the *destination* of aliases. Aliasing occurs before [<Directory>](#) sections are checked, so only the destination of aliases are affected. (Note however [<Location>](#) sections are run through once before aliases are performed, so they will apply.)

See also [ScriptAlias](#).

AliasMatch

Syntax: `AliasMatch regex file-path|directory-path`

Context: server config, virtual host

Status: Base

Module: mod_alias

Compatibility: Available in Apache 1.3 and later

This directive is equivalent to [Alias](#), but makes use of standard regular expressions, instead of simple prefix matching. The supplied regular expression is matched against the URL-path, and if it matches, the server will substitute any parenthesized matches into the given string and use it as a filename. For example, to activate the `/icons` directory, one might use:

```
AliasMatch ^/icons(.*) /usr/local/apache/icons$1
```

Redirect directive

Syntax: `Redirect [status] URL-path URL`

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_alias

Compatibility: The directory and .htaccess context's are only available in versions 1.1 and later. The *status* argument is only available in Apache 1.2 or later.

The Redirect directive maps an old URL into a new one. The new URL is returned to the client which attempts to fetch it again with the new address. *URL-path* a (%-decoded) path; any requests for documents beginning with this path will be returned a redirect error to a new (%-encoded) URL beginning with *URL*.

Example:

```
Redirect /service http://foo2.bar.com/service
```

If the client requests `http://myserver/service/foo.txt`, it will be told to access `http://foo2.bar.com/service/foo.txt` instead.

Note: Redirect directives take precedence over [Alias](#) and [ScriptAlias](#) directives, irrespective of their ordering in the configuration file. Also, *URL-path* must be an absolute path, not a relative path, even when used with .htaccess files or inside of [<Directory>](#) sections.

If no *status* argument is given, the redirect will be "temporary" (HTTP status 302). This indicates to the client that the resource has moved temporarily. The *status* argument can be used to return other HTTP status codes:

permanent

Returns a permanent redirect status (301) indicating that the resource has moved permanently.

temp

Returns a temporary redirect status (302). This is the default.

seeother

Returns a "See Other" status (303) indicating that the resource has been replaced.

gone

Returns a "Gone" status (410) indicating that the resource has been permanently removed. When this status is used the *url* argument should be omitted.

Other status codes can be returned by giving the numeric status code as the value of *status*. If the status is between 300 and 399, the *url* argument must be present, otherwise it must be omitted. Note that the status must be known to the Apache code (see the function `send_error_response` in `http_protocol.c`).

RedirectMatch

Syntax: `RedirectMatch [status] regex URL`

Context: server config, virtual host, directory, `.htaccess`

Override: FileInfo

Status: Base

Module: mod_alias

Compatibility: Available in Apache 1.3 and later

This directive is equivalent to [Redirect](#), but makes use of standard regular expressions, instead of simple prefix matching. The supplied regular expression is matched against the URL-path, and if it matches, the server will substitute any parenthesized matches into the given string and use it as a filename. For example, to redirect all GIF files to like-named JPEG files on another server, one might use:

```
RedirectMatch (.*)\.gif$ http://www.anotherserver.com$1.jpg
```

RedirectTemp directive

Syntax: `RedirectTemp URL-path URL`

Context: server config, virtual host, directory, `.htaccess`

Override: FileInfo

Status: Base

Module: mod_alias

Compatibility: This directive is only available in Apache 1.2 and later

This directive makes the client know that the Redirect is only temporary (status 302). Exactly equivalent to `Redirect temp`.

RedirectPermanent directive

Syntax: `RedirectPermanent URL-path URL`

Context: server config, virtual host, directory, `.htaccess`

Override: FileInfo

Status: Base

Module: mod_alias

Compatibility: This directive is only available in Apache 1.2 and later

This directive makes the client know that the Redirect is permanent (status 301). Exactly equivalent to `Redirect permanent`.

ScriptAlias directive

Syntax: `ScriptAlias URL-path file-path|directory-path`

Context: server config, virtual host

Status: Base

Module: mod_alias

The ScriptAlias directive has the same behavior as the [Alias](#) directive, except that in addition it marks the target directory as containing CGI scripts. URLs with a (%-decoded) path beginning with *URL-path* will be mapped to scripts beginning with the second argument which is a full pathname in the local filesystem.

Example:

```
ScriptAlias /cgi-bin/ /web/cgi-bin/
```

A request for `http://myserver/cgi-bin/foo` would cause the server to run the script `/web/cgi-bin/foo`.

ScriptAliasMatch

Syntax: `ScriptAliasMatch regex file-path|directory-path`

Context: server config, virtual host

Status: Base

Module: mod_alias

Compatibility: Available in Apache 1.3 and later

This directive is equivalent to [ScriptAlias](#), but makes use of standard regular expressions, instead of simple prefix matching. The supplied regular expression is matched against the URL-path, and if it matches, the server will substitute any parenthesized matches into the given string and use it as a filename. For example, to activate the standard `/cgi-bin`, one might use:

```
ScriptAliasMatch ^/cgi-bin(.*) /usr/local/apache/cgi-bin$1
```

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_asis

This module provides for sending files which contain their own HTTP headers.

Status: Base

Source File: mod_asis.c

Module Identifier: asis_module

Summary

This module provides the handler `send-as-is` which causes Apache to send the document without adding most of the usual HTTP headers.

This can be used to send any kind of data from the server, including redirects and other special HTTP responses, without requiring a `cgi-script` or an `nph` script.

For historical reasons, this module will also process any file with the mime type `httpd/send-as-is`.

Directives

This module provides no directives.

Usage

In the server configuration file, associate files with the `send-as-is` handler *e.g.*

```
AddHandler send-as-is asis
```

The contents of any file with a `.asis` extension will then be sent by Apache to the client with almost no changes. Clients will need HTTP headers to be attached, so do not forget them. A `Status:` header is also required; the data should be the 3-digit HTTP response code, followed by a textual message.

Here's an example of a file whose contents are sent *as is* so as to tell the client that a file has redirected.

```
Status: 301 Now where did I leave that URL
Location: http://xyz.abc.com/foo/bar.html
Content-type: text/html

<HTML>
<HEAD>
<TITLE>Lame excuses'R'us</TITLE>
</HEAD>
<BODY>
<H1>Fred's exceptionally wonderful page has moved to
<A HREF="http://xyz.abc.com/foo/bar.html">Joe's</A> site.
</H1>
</BODY>
</HTML>
```

Notes: the server always adds a Date: and Server: header to the data returned to the client, so these should not be included in the file. The server does *not* add a Last-Modified header; it probably should.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_auth

This module provides for user authentication using text files.

Status: Base

Source File: mod_auth.c

Module Identifier: auth_module

Summary

This module allows the use of HTTP Basic Authentication to restrict access by looking up users in plain text password and group files. Similar functionality and greater scalability is provided by [mod_auth_dbm](#) and [mod_auth_db](#). HTTP Digest Authentication is provided by [mod_auth_digest](#).

Note that these credential-based security mechanisms are only as strong as your Web server's security. As a rule, they are *not* as strong as the operating system's own security system.

Directives

- [AuthGroupFile](#)
- [AuthUserFile](#)
- [AuthAuthoritative](#)

See also: [require](#), [satisfy](#), and [mod_auth require keywords](#).

mod_auth Require Keywords

The mod_auth module supports the following keywords that can be given to the [Require](#) directive:

`user username [...]`

The supplied username and password must be in the [AuthUserFile](#) database, and the username must also be one of those listed on the Require directive.

`group groupname [...]`

The supplied username and password must be in the [AuthUserFile](#) database, and the username must also be a member of one of the named groups in the [AuthGroupFile](#) database.

`valid-user`

The supplied username and password must be in the [AuthUserFile](#) database. Any valid username from that file will be allowed.

`file-owner`

[Available after Apache 1.3.20] The supplied username and password must be in the [AuthUserFile](#) database, and the username must also match the system's name for the owner of the file being requested. That is, if the operating system say

the requested file is owned by jones, then the username used to access it through the Web must be jones as well.

file-group

[Available after Apache 1.3.20] The supplied username and password must be in the [AuthUserFile](#) database, the name of the group that owns the file must be in the [AuthGroupFile](#) database, and the username must be a member of that group. For example, if the operating system says the requested file is owned by group accounts, the group accounts must be in the AuthGroupFile database and the username used in the request must be a member of that group.

Example of Require file-owner

Consider a multi-user system running the Apache Web server, with each user having his or her own files in ~/public_html/private. Assuming that there is a single AuthUserFile database that lists all of their usernames, and that their Web usernames match the ones that actually own the files on the server, then the following stanza would allow only the user himself access to his own files. User jones would not be allowed to access files in /home/smith/public_html/private unless they were owned by jones instead of smith.

```
<Directory /home/*/public_html/private>
  AuthType Basic
  AuthName MyPrivateFile
  AuthUserFile /usr/local/apache/etc/.htpasswd-allusers
  Satisfy All
  Require file-owner
</Directory>
```

AuthGroupFile directive

Syntax: AuthGroupFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Base

Module: mod_auth

The AuthGroupFile directive sets the name of a textual file containing the list of user groups for user authentication. *File-path* is the path to the group file. If it is not absolute (*i.e.*, if it doesn't begin with a slash), it is treated as relative to the ServerRoot.

Each line of the group file contains a groupname followed by a colon, followed by the member usernames separated by spaces. Example:

```
mygroup: bob joe anne
```

Note that searching large text files is *very* inefficient; [AuthDBMGroupFile](#) should be used instead.

Security: make sure that the AuthGroupFile is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients will be able to download the AuthGroupFile.

See also [AuthName](#), [AuthType](#) and [AuthUserFile](#).

AuthUserFile directive

Syntax: AuthUserFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Base

Module: mod_auth

The AuthUserFile directive sets the name of a textual file containing the list of users and passwords for user authentication. *File-path* is the path to the user file. If it is not absolute (*i.e.*, if it doesn't begin with a slash), it is treated as relative to the ServerRoot.

Each line of the user file contains a username followed by a colon, followed by the `crypt()` encrypted password. The behavior of multiple occurrences of the same user is undefined.

The utility [htpasswd](#) which is installed as part of the binary distribution, or which can be found in `src/support`, is used to maintain this password file. See the man page for more details. In short

```
htpasswd -c Filename username
Create a password file 'Filename' with 'username' as the initial ID. It will prompt for the password. htpasswd
Filename username2
Adds or modifies in password file 'Filename' the 'username'.
```

Note that searching large text files is *very* inefficient; [AuthDBMUserFile](#) should be used instead.

Security:

Make sure that the AuthUserFile is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients may be able to download the AuthUserFile.

Also be aware that null usernames are permitted, and null passwords as well (through Apache 1.3.20). If your AuthUserFile includes a line containing only a colon (:), a `Require valid-user` will allow access if both the username and password in the credentials are omitted.

See also [AuthName](#), [AuthType](#) and [AuthGroupFile](#).

AuthAuthoritative directive

Syntax: AuthAuthoritative on|off**Default:** AuthAuthoritative on**Context:** directory, .htaccess**Override:** AuthConfig**Status:** Base**Module:** mod_auth

Setting the AuthAuthoritative directive explicitly to **'off'** allows for both authentication and authorization to be passed on to lower level modules (as defined in the `Configuration` and `modules.c` files) if there is **no userID** or **rule** matching the supplied userID. If there is a userID and/or rule specified; the usual password and access checks will be applied and a failure will give an Authorization Required reply.

So if a userID appears in the database of more than one module; or if a valid `Require` directive applies to more than one module; then the first module will verify the credentials; and no access is passed on; regardless of the AuthAuthoritative setting.

A common use for this is in conjunction with one of the database modules; such as [mod_auth_db.c](#), [mod_auth_dbm.c](#), [mod_auth_mysql.c](#), and [mod_auth_anon.c](#). These modules supply the bulk of the user credential checking; but a few (administrator) related accesses fall through to a lower level with a well protected AuthUserFile.

Default: By default; control is not passed on; and an unknown userID or rule will result in an Authorization Required reply. Not setting it thus keeps the system secure; and forces an NCSA compliant behavior.

Security: Do consider the implications of allowing a user to allow fall-through in his `.htaccess` file; and verify that this is really what you want; Generally it is easier to just secure a single `htpasswd` file, than it is to secure a database such as mSQL. Make sure that the AuthUserFile is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients will be able to download the AuthUserFile.

See also [AuthName](#), [AuthType](#) and [AuthGroupFile](#).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_auth_anon

This module allows "anonymous" user access to authenticated areas.

Status: Extension

Source File: mod_auth_anon.c

Module Identifier: anon_auth_module

Compatibility: Available in Apache 1.1 and later.

Summary

This module does access control in a manner similar to anonymous-ftp sites; *i.e.* have a 'magic' user id 'anonymous' and the email address as a password. These email addresses can be logged.

Combined with other (database) access control methods, this allows for effective user tracking and customization according to a user profile while still keeping the site open for 'unregistered' users. One advantage of using Auth-based user tracking is that, unlike magic-cookies and funny URL pre/postfixes, it is completely browser independent and it allows users to share URLs.

Directives

- [Anonymous](#)
- [Anonymous_Authoritative](#)
- [Anonymous_LogEmail](#)
- [Anonymous_MustGiveEmail](#)
- [Anonymous_NoUserID](#)
- [Anonymous_VerifyEmail](#)

Example

The example below (when combined with the Auth directives of a htpasswd-file based (or GDM, mSQL *etc.*) base access control system allows users in as 'guests' with the following properties:

- It insists that the user enters a userID. (`Anonymous_NoUserId`)
- It insists that the user enters a password. (`Anonymous_MustGiveEmail`)
- The password entered must be a valid email address, *ie.* contain at least one '@' and a '.'. (`Anonymous_VerifyEmail`)
- The userID must be one of `anonymous guest www test welcome` and comparison is **not** case sensitive.
- And the Email addresses entered in the passwd field are logged to the error log file (`Anonymous_LogEmail`)

Excerpt of access.conf:

```
Anonymous_NoUserId      off
Anonymous_MustGiveEmail on
Anonymous_VerifyEmail   on
```

```
Anonymous_LogEmail      on
Anonymous               anonymous guest www test welcome

AuthName                "Use 'anonymous' & Email address for guest entry"
AuthType                basic

# An AuthUserFile/AuthDBUserFile/AuthDBMUserFile
# directive must be specified, or use
# Anonymous_Authoritative for public access.
# In the .htaccess for the public directory, add:
<Files *>
Order Deny,Allow
Allow from all

Require valid-user
</Files>
```

Anonymous directive

Syntax: `Anonymous user [user] ...`

Default: none

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_anon

A list of one or more 'magic' userIDs which are allowed access without password verification. The userIDs are space separated. It is possible to use the ' and " quotes to allow a space in a userID as well as the \ escape character.

Please note that the comparison is **case-IN-sensitive**.

I strongly suggest that the magic username 'anonymous' is always one of the allowed userIDs.

Example:

```
Anonymous anonymous "Not Registered" 'I don\'t know'
```

This would allow the user to enter without password verification by using the userID's 'anonymous', 'AnonyMous','Not Registered' and 'I Don't Know'.

Anonymous_Authoritative directive

Syntax: `Anonymous_Authoritative on|off`

Default: `Anonymous_Authoritative off`

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_anon

When set 'on', there is no fall-through to other authorization methods. So if a userID does not match the values specified in the Anonymous directive, access is denied.

Be sure you know what you are doing when you decide to switch it on. And remember that it is the linking order of the modules (in the Configuration / Make file) which details the order in which the Authorization modules are queried.

Anonymous_LogEmail directive

Syntax: Anonymous_LogEmail on|off

Default: Anonymous_LogEmail on

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_anon

When set 'on', the default, the 'password' entered (which hopefully contains a sensible email address) is logged in the error log.

Anonymous_MustGiveEmail directive

Syntax: Anonymous_MustGiveEmail on|off

Default: Anonymous_MustGiveEmail on

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_anon

Specifies whether the user must specify an email address as the password. This prohibits blank passwords.

Anonymous_NoUserID directive

Syntax: Anonymous_NoUserID on|off

Default: Anonymous_NoUserID off

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_anon

When set 'on', users can leave the userID (and perhaps the password field) empty. This can be very convenient for MS-Explorer users who can just hit return or click directly on the OK button; which seems a natural reaction.

Anonymous_VerifyEmail directive

Syntax: Anonymous_VerifyEmail on|off

Default: Anonymous_VerifyEmail off

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_anon

When set 'on' the 'password' entered is checked for at least one '@' and a '.' to encourage users to enter valid email addresses (see the above Auth_LogEmail).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_auth_db

This module provides for user authentication using Berkeley DB files.

Status: Extension

Source File: mod_auth_db.c

Module Identifier: db_auth_module

Compatibility: Available in Apache 1.1 and later.

Summary

This module provides an alternative to [DBM](#) files for those systems which support DB and not DBM. It is only available in Apache 1.1 and later.

On some BSD systems (*e.g.*, FreeBSD and NetBSD) dbm is automatically mapped to Berkeley DB. You can use either [mod_auth_dbm](#) or mod_auth_db. The latter makes it more obvious that it's Berkeley DB. On other platforms where you want to use the DB library you usually have to install it first. See <http://www.sleepycat.com/> for the distribution. The interface this module uses is the one from DB version 1.85 and 1.86, but DB version 2.x can also be used when compatibility mode is enabled.

Directives

- [AuthDBGroupFile](#)
- [AuthDBUserFile](#)
- [AuthDBAuthoritative](#)

See also: [satisfy](#) and [require](#).

AuthDBGroupFile directive

Syntax: AuthDBGroupFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_db

The AuthDBGroupFile directive sets the name of a DB file containing the list of user groups for user authentication. *File-path* is the absolute path to the group file.

The group file is keyed on the username. The value for a user is a comma-separated list of the groups to which the users belongs. There must be no whitespace within the value, and it must never contain any colons.

Security: make sure that the AuthDBGroupFile is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients will be able to download the AuthDBGroupFile unless otherwise protected.

Combining Group and Password DB files: In some cases it is easier to manage a single database which contains both the password and group details for each user. This simplifies any support programs that need to be written: they now only have to deal with writing to and locking a single DBM file. This can be accomplished by first setting the group and password files to point to the same DB file:

```
AuthDBGroupFile /www/userbase
AuthDBUserFile /www/userbase
```

The key for the single DB record is the username. The value consists of

```
Unix Crypt-ed Password : List of Groups [ : (ignored) ]
```

The password section contains the Unix crypt() password as before. This is followed by a colon and the comma separated list of groups. Other data may optionally be left in the DB file after another colon; it is ignored by the authentication module.

See also [AuthName](#), [AuthType](#) and [AuthDBUserFile](#).

AuthDBUserFile directive

Syntax: AuthDBUserFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_db

The AuthDBUserFile directive sets the name of a DB file containing the list of users and passwords for user authentication. *File-path* is the absolute path to the user file.

The user file is keyed on the username. The value for a user is the crypt() encrypted password, optionally followed by a colon and arbitrary data. The colon and the data following it will be ignored by the server.

Security: make sure that the AuthDBUserFile is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients will be able to download the AuthDBUserFile.

Important compatibility note: The implementation of "dbmopen" in the apache modules reads the string length of the hashed values from the DB data structures, rather than relying upon the string being NULL-appended. Some applications, such as the Netscape web server, rely upon the string being NULL-appended, so if you are having trouble using DB files interchangeably between applications this may be a part of the problem.

A perl script called `href=" ../programs/dbmmanage.html">dbmmanage` is included with Apache. This program can be used to create and update DB format password files for use with this module.

See also [AuthName](#), [AuthType](#) and [AuthDBGroupFile](#).

AuthDBAuthoritative directive

Syntax: AuthDBAuthoritative on|off

Default: AuthDBAuthoritative on

Context: directory, .htaccess

Override: AuthConfig

Status: Base

Module: mod_auth

Setting the AuthDBAuthoritative directive explicitly to 'off' allows for both authentication and authorization to be passed on to lower level modules (as defined in the Configuration and modules.c file if there is **no userID** or **rule** matching the supplied userID. If there is a userID and/or rule specified; the usual password and access checks will be applied and a failure will give an Authorization Required reply.

So if a userID appears in the database of more than one module; or if a valid Require directive applies to more than one module;

then the first module will verify the credentials; and no access is passed on; regardless of the AuthAuthoritative setting.

A common use for this is in conjunction with one of the basic auth modules; such as [mod_auth.c](#). Whereas this DB module supplies the bulk of the user credential checking; a few (administrator) related accesses fall through to a lower level with a well protected .htpasswd file.

By default, control is not passed on and an unknown userID or rule will result in an Authorization Required reply. Not setting it thus keeps the system secure and forces an NCSA compliant behavior.

Security: Do consider the implications of allowing a user to allow fall-through in his .htaccess file; and verify that this is really what you want; Generally it is easier to just secure a single .htpasswd file, than it is to secure a database which might have more access interfaces.

See also [AuthName](#), [AuthType](#) and [AuthDBGroupFile](#).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_auth_dbm

This module provides for user authentication using DBM files.

Status: Extension

Source File: mod_auth_dbm.c

Module Identifier: dbm_auth_module

Summary

This module provides for HTTP Basic Authentication, where the usernames and passwords are stored in DBM type database files. It is an alternative to the plain text password files provided by [mod_auth](#) and the Berkely DB password files provided by [mod_auth_db](#).

Directives

- [AuthDBMGroupFile](#)
- [AuthDBMUserFile](#)
- [AuthDBMAuthoritative](#)

See also: [Satisfy](#) and [Require](#).

AuthDBMGroupFile

Syntax: AuthDBMGroupFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_dbm

The AuthDBMGroupFile directive sets the name of a DBM file containing the list of user groups for user authentication. *File-path* is the absolute path to the group file.

The group file is keyed on the username. The value for a user is a comma-separated list of the groups to which the users belongs. There must be no whitespace within the value, and it must never contain any colons.

Security: make sure that the AuthDBMGroupFile is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients will be able to download the AuthDBMGroupFile unless otherwise protected.

Combining Group and Password DBM files: In some cases it is easier to manage a single database which contains both the password and group details for each user. This simplifies any support programs that need to be written: they now only have to deal with writing to and locking a single DBM file. This can be accomplished by first setting the group and password files to point to the same DBM:

Apache module mod_auth_dbm

```
AuthDBMGroupFile /www/userbase
```

```
AuthDBMUserFile /www/userbase
```

The key for the single DBM is the username. The value consists of

```
Unix Crypt-ed Password : List of Groups [ : (ignored) ]
```

The password section contains the Unix crypt() password as before. This is followed by a colon and the comma separated list of groups. Other data may optionally be left in the DBM file after another colon; it is ignored by the authentication module. This is what www.telescope.org uses for its combined password and group database.

See also [AuthName](#), [AuthType](#) and [AuthDBMUserFile](#).

AuthDBMUserFile

Syntax: AuthDBMUserFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_dbm

The AuthDBMUserFile directive sets the name of a DBM file containing the list of users and passwords for user authentication. *File-path* is the absolute path to the user file.

The user file is keyed on the username. The value for a user is the crypt() encrypted password, optionally followed by a colon and arbitrary data. The colon and the data following it will be ignored by the server.

Security: make sure that the AuthDBMUserFile is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients will be able to download the AuthDBMUserFile.

Important compatibility note: The implementation of "dbmopen" in the apache modules reads the string length of the hashed values from the DBM data structures, rather than relying upon the string being NULL-appended. Some applications, such as the Netscape web server, rely upon the string being NULL-appended, so if you are having trouble using DBM files interchangeably between applications this may be a part of the problem.

A perl script called [dbmmanage](#) is included with Apache. This program can be used to create and update DBM format password files for use with this module.

See also [AuthName](#), [AuthType](#) and [AuthDBMGroupFile](#).

AuthDBMAuthoritative

Syntax: AuthDBMAuthoritative on|off

Default: AuthDBMAuthoritative on

Context: directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_auth_dbm

Setting the AuthDBMAuthoritative directive explicitly to '**off**' allows for both authentication and authorization to be passed on to lower level modules (as defined in the Configuration and modules.c file if there is **no userID** or **rule** matching the supplied userID. If there is a userID and/or rule specified; the usual password and access checks will be applied and a failure will give an Authorization Required reply.

So if a userID appears in the database of more than one module; or if a valid Require directive applies to more than one module; then the first module will verify the credentials; and no access is passed on; regardless of the AuthAuthoritative setting.

A common use for this is in conjunction with one of the basic auth modules; such as [mod_auth.c](#). Whereas this DBM module

supplies the bulk of the user credential checking; a few (administrator) related accesses fall through to a lower level with a well protected .htpasswd file.

By default, control is not passed on and an unknown userID or rule will result in an Authorization Required reply. Not setting it thus keeps the system secure and forces an NCSA compliant behavior.

Security: Do consider the implications of allowing a user to allow fall-through in his .htaccess file; and verify that this is really what you want; Generally it is easier to just secure a single .htpasswd file, than it is to secure a database which might have more access interfaces.

See also [AuthName](#), [AuthType](#) and [AuthDBMGroupFile](#).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_auth_digest

This module provides for user authentication using MD5 Digest Authentication.

Status: Experimental

Source File: mod_auth_digest.c

Module Identifier: digest_auth_module

Compatibility: Available in Apache 1.3.8 and later.

Summary

This is an updated version of [mod_digest](#). However, it has not been extensively tested and is therefore marked experimental. If you use this module, you must make sure to *not* use mod_digest (because they share some of the same configuration directives).

Digest authentication is described in [RFC 2617](#).

Directives

- [AuthDigestFile](#)
- [AuthDigestGroupFile](#)
- [AuthDigestQop](#)
- [AuthDigestNonceLifetime](#)
- [AuthDigestNonceFormat](#)
- [AuthDigestNcCheck](#)
- [AuthDigestAlgorithm](#)
- [AuthDigestDomain](#)

See also: [Require](#) and [Satisfy](#).

Using Digest Authentication

Using MD5 Digest authentication is very simple. Simply set up authentication normally, using "AuthType Digest" and "AuthDigestFile" instead of the normal "AuthType Basic" and "AuthUserFile"; also, replace any "AuthGroupFile" with "AuthDigestGroupFile". Then add a "AuthDigestDomain" directive containing at least the root URI(s) for this protection space. Example:

```
<Location /private/>
AuthType Digest
AuthName "private area"
AuthDigestDomain /private/ http://mirror.my.dom/private2/
AuthDigestFile /web/auth/.digest_pw
Require valid-user
</Location>
```

Note: MD5 authentication provides a more secure password system than Basic authentication, but only works with supporting browsers. As of this writing (October 2001), the only major browsers which support digest authentication are [Opera 4.0](#), [MS Internet Explorer 5.0](#) and [Amaya](#). Therefore, we do not yet recommend using this feature on a large Internet site. However, for personal and intra-net use, where browser users can be controlled, it is ideal.

AuthDigestFile directive

Syntax: AuthDigestFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Experimental

Module: mod_auth_digest

The AuthDigestFile directive sets the name of a textual file containing the list of users and encoded passwords for digest authentication. *File-path* is the absolute path to the user file.

The digest file uses a special format. Files in this format can be created using the [htdigest](#) utility found in the support/ subdirectory of the Apache distribution.

AuthDigestGroupFile directive

Syntax: AuthDigestGroupFile *file-path*

Context: directory, .htaccess

Override: AuthConfig

Status: Experimental

Module: mod_auth_digest

Compatibility: Available in Apache 1.3.8 and later

The AuthDigestGroupFile directive sets the name of a textual file containing the list of groups and their members (user names). *File-path* is the absolute path to the group file.

Each line of the group file contains a groupname followed by a colon, followed by the member usernames separated by spaces. Example:

```
mygroup: bob joe anne
```

Note that searching large text files is *very* inefficient.

Security: make sure that the AuthGroupFile is stored outside the document tree of the web-server; do *not* put it in the directory that it protects. Otherwise, clients will be able to download the AuthGroupFile.

AuthDigestQop directive

Syntax: AuthDigestQop none|auth|auth-int [auth|auth-int]

Default: AuthDigestQop auth

Context: directory, .htaccess

Override: AuthConfig

Status: Experimental

Module: mod_auth_digest

Compatibility: Available in Apache 1.3.8 and later

The AuthDigestQop directive determines the quality-of-protection to use. *auth* will only do authentication (username/password); *auth-int* is authentication plus integrity checking (an MD5 hash of the entity is also computed and checked); *none* will cause the

module to use the old RFC-2069 digest algorithm (which does not include integrity checking). Both *auth* and *auth-int* may be specified, in which the case the browser will choose which of these to use. *none* should only be used if the browser for some reason does not like the challenge it receives otherwise.

***auth-int* is not implemented yet.**

AuthDigestNonceLifetime directive

Syntax: AuthDigestNonceLifetime *seconds*

Default: AuthDigestNonceLifetime 300

Context: directory, .htaccess

Override: AuthConfig

Status: Experimental

Module: mod_auth_digest

Compatibility: Available in Apache 1.3.8 and later

The AuthDigestNonceLifetime directive controls how long the server nonce is valid. When the client contacts the server using an expired nonce the server will send back a 401 with `stale=true`. If *seconds* is greater than 0 then it specifies the amount of time for which the nonce is valid; this should probably never be set to less than 10 seconds. If *seconds* is less than 0 then the nonce never expires.

AuthDigestNonceFormat directive

Syntax: AuthDigestNonceFormat ???

Default: AuthDigestNonceFormat ???

Context: directory, .htaccess

Override: AuthConfig

Status: Experimental

Module: mod_auth_digest

Compatibility: Available in Apache 1.3.8 and later

Not implemented yet.

AuthDigestNcCheck directive

Syntax: AuthDigestNcCheck On|Off

Default: AuthDigestNcCheck Off

Context: server config

Override: *Not applicable*

Status: Experimental

Module: mod_auth_digest

Compatibility: Available in Apache 1.3.8 and later

Not implemented yet.

AuthDigestAlgorithm directive

Syntax: AuthDigestAlgorithm MD5|MD5-sess

Default: AuthDigestAlgorithm MD5

Context: directory, .htaccess

Override: AuthConfig

Status: Experimental

Module: mod_auth_digest

Compatibility: Available in Apache 1.3.8 and later

The AuthDigestAlgorithm directive selects the algorithm used to calculate the challenge and response hashes.

MD5-sess is not correctly implemented yet.

AuthDigestDomain directive

Syntax: AuthDigestDomain *URI* [*URI*] ...

Context: directory, .htaccess

Override: AuthConfig

Status: Experimental

Module: mod_auth_digest

Compatibility: Available in Apache 1.3.8 and later

The AuthDigestDomain directive allows you to specify one or more URIs which are in the same protection space (i.e. use the same realm and username/password info). The specified URIs are prefixes, i.e. the client will assume that all URIs "below" these are also protected by the same username/password. The URIs may be either absolute URIs (i.e. including a scheme, host, port, etc) or relative URIs.

This directive *should* always be specified and contain at least the (set of) root URI(s) for this space. Omitting to do so will cause the client to send the Authorization header for *every request* sent to this server. Apart from increasing the size of the request, it may also have a detrimental effect on performance if "AuthDigestNcCheck" is on.

The URIs specified can also point to different servers, in which case clients (which understand this) will then share username/password info across multiple servers without prompting the user each time.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_autoindex

This module provides for automatic directory indexing.

Status: Base

Source File: mod_autoindex.c

Module Identifier: autoindex_module

Summary

The index of a directory can come from one of two sources:

- A file written by the user, typically called `index.html`. The [DirectoryIndex](#) directive sets the name of this file. This is controlled by [mod_dir](#).
- Otherwise, a listing generated by the server. The other directives control the format of this listing. The [AddIcon](#), [AddIconByEncoding](#) and [AddIconByType](#) are used to set a list of icons to display for various file types; for each file listed, the first icon listed that matches the file is displayed. These are controlled by `mod_autoindex`.

The two functions are separated so that you can completely remove (or replace) automatic index generation should you want to.

Automatic index generation is enabled with using `Options +Indexes`. See the [Options](#) directive for more details.

If [FancyIndexing](#) is enabled, or the `FancyIndexing` keyword is present on the [IndexOptions](#) directive, the column headers are links that control the order of the display. If you select a header link, the listing will be regenerated, sorted by the values in that column. Selecting the same header repeatedly toggles between ascending and descending order.

Note that when the display is sorted by "Size", it's the *actual* size of the files that's used, not the displayed value - so a 1010-byte file will always be displayed before a 1011-byte file (if in ascending order) even though they both are shown as "1K".

Directives

- [AddAlt](#)
- [AddAltByEncoding](#)
- [AddAltByType](#)
- [AddDescription](#)
- [AddIcon](#)
- [AddIconByEncoding](#)
- [AddIconByType](#)
- [DefaultIcon](#)
- [FancyIndexing](#)
- [HeaderName](#)
- [IndexIgnore](#)

- [IndexOptions](#)
- [IndexOrderDefault](#)
- [ReadmeName](#)

See also: [Options](#) and [DirectoryIndex](#).

Autoindex Request Query Arguments

The column sorting headers themselves are self-referencing hyperlinks that add the sort query options to reorder the directory listing. The query options are of the form X=Y, where X is one of N (file Name), M (file last Modified date), S (file Size, or D (file Description), and Y is one of A (Ascending) or D (Descending).

When options other than the file name are used as the sorting key, the secondary key is always the file name. (When the file name is used to sort by, there is no need of a secondary sort key, since file names are guaranteed to be unique, and so the sort order is unambiguous.)

Example:

If the URL <http://your.server.name/foo/> produces a directory index, then the following URLs will produce different sort orders:

- <http://your.server.name/foo/?M=D> sorts the directory by last modified date, descending.
- <http://your.server.name/foo/?D=A> sorts the directory by file description, ascending.
- <http://your.server.name/foo/?S=A> sorts the directory by file size, ascending.

See [IndexOrderDefault](#) to set the default directory ordering.

Note also that when the directory listing is ordered in one direction (ascending or descending) by a particular column, the link at the top of that column then reverses, to allow sorting in the opposite direction by that same column.

AddAlt directive

Syntax: AddAlt *string file [file] ...*

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

This sets the alternate text to display for a file, instead of an icon, for [FancyIndexing](#). *File* is a file extension, partial filename, wild-card expression or full filename for files to describe. *String* is enclosed in double quotes ("). This alternate text is displayed if the client is image-incapable or has image loading disabled.

Examples:

```
AddAlt "PDF" *.pdf
AddAlt "Compressed" *.gz *.zip *.Z
```

AddAltByEncoding directive

Syntax: AddAltByEncoding *string MIME-encoding [MIME-encoding] ...*

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

This sets the alternate text to display for a file, instead of an icon, for [FancyIndexing](#). *MIME-encoding* is a valid content-encoding, such as x-compress. *String* is enclosed in double quotes ("). This alternate text is displayed if the client is image-incapable or has image loading disabled.

Example:

```
AddAltByEncoding "gzip" x-gzip
```

AddAltByType directive

Syntax: AddAltByType *string* *MIME-type* [*MIME-type*] ...

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

This sets the alternate text to display for a file, instead of an icon, for [FancyIndexing](#). *MIME-type* is a valid content-type, such as text/html. *String* is enclosed in double quotes ("). This alternate text is displayed if the client is image-incapable or has image loading disabled.

Example:

```
AddAltByType "TXT" text/plain
```

AddDescription directive

Syntax: AddDescription *string* *file* [*file*] ...

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

This sets the description to display for a file, for [FancyIndexing](#). *File* is a file extension, partial filename, wild-card expression or full filename for files to describe. *String* is enclosed in double quotes ("). Example:

```
AddDescription "The planet Mars" /web/pics/mars.gif
```

The description field is 23 bytes wide. 7 more bytes may be added if the directory is covered by an `IndexOptions SuppressSize`, and 19 bytes may be added if `IndexOptions SuppressLastModified` is in effect. The widest this column can be is therefore 49 bytes.

As of Apache 1.3.10, the [DescriptionWidth](#) `IndexOptions` keyword allows you to adjust this width to any arbitrary size.

Caution: Descriptive text defined with `AddDescription` may contain HTML markup, such as tags and character entities. If the width of the description column should happen to truncate a tagged element (such as cutting off the end of a bolded phrase), the results may affect the rest of the directory listing.

AddIcon directive

Syntax: AddIcon *icon name* [*name*] ...

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

This sets the icon to display next to a file ending in *name* for [FancyIndexing](#). *Icon* is either a (%-escaped) relative URL to the icon, or of the format (*alttext,url*) where *alttext* is the text tag given for an icon for non-graphical browsers.

Name is either `^^DIRECTORY^^` for directories, `^^BLANKICON^^` for blank lines (to format the list correctly), a file extension, a wildcard expression, a partial filename or a complete filename. Examples:

```
AddIcon (IMG,/icons/image2.gif) .gif .jpg .png
AddIcon /icons/dir.gif ^^DIRECTORY^^
AddIcon /icons/backup.gif *~
```

[AddIconByType](#) should be used in preference to AddIcon, when possible.

AddIconByEncoding directive

Syntax: AddIconByEncoding *icon MIME-encoding* [*MIME-encoding*] ...

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

This sets the icon to display next to files with *MIME-encoding* for [FancyIndexing](#). *Icon* is either a (%-escaped) relative URL to the icon, or of the format (*alttext,url*) where *alttext* is the text tag given for an icon for non-graphical browsers.

Mime-encoding is a wildcard expression matching required the content-encoding. Examples:

```
AddIconByEncoding /icons/compressed.gif x-compress
```

AddIconByType directive

Syntax: AddIconByType *icon MIME-type* [*MIME-type*] ...

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

This sets the icon to display next to files of type *MIME-type* for [FancyIndexing](#). *Icon* is either a (%-escaped) relative URL to the icon, or of the format (*alttext,url*) where *alttext* is the text tag given for an icon for non-graphical browsers.

Mime-type is a wildcard expression matching required the mime types. Examples:

```
AddIconByType (IMG,/icons/image3.gif) image/*
```

DefaultIcon directive

Syntax: `DefaultIcon url`

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

The `DefaultIcon` directive sets the icon to display for files when no specific icon is known, for [FancyIndexing](#). *Url* is a (%-escaped) relative URL to the icon. Examples:

```
DefaultIcon /icon/unknown.xbm
```

FancyIndexing directive

Syntax: `FancyIndexing on|off`

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

The `FancyIndexing` directive sets the `FancyIndexing` option for a directory. The [IndexOptions](#) directive should be used in preference.

Note that in versions of Apache prior to 1.3.2, the `FancyIndexing` and `IndexOptions` directives will override each other. You should use `IndexOptions FancyIndexing` in preference to the standalone `FancyIndexing` directive. As of Apache 1.3.2, a standalone `FancyIndexing` directive is combined with any `IndexOptions` directive already specified for the current scope.

HeaderName directive

Syntax: `HeaderName filename`

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

Compatibility: some features only available after 1.3.6; see text

The `HeaderName` directive sets the name of the file that will be inserted at the top of the index listing. *Filename* is the name of the file to include.

Apache 1.3.6 and earlier: The module first attempts to include *filename*.html as an HTML document, otherwise it will try to include *filename* as plain text. *Filename* is treated as a filesystem path relative to the directory being indexed. In no case is SSI processing done. Example:

```
HeaderName HEADER
```

when indexing the directory `/web`, the server will first look for the HTML file `/web/HEADER.html` and include it if found, otherwise it will include the plain text file `/web/HEADER`, if it exists.

Apache versions after 1.3.6: *Filename* is treated as a URI path relative to the one used to access the directory being indexed, and must resolve to a document with a major content type of "text" (e.g., `text/html`, `text/plain`, etc.). This means that *filename* may refer to a CGI script if the script's actual file type (as opposed to its output) is marked as `text/html` such as with a directive like:

```
AddType text/html .cgi
```

[Content negotiation](#) will be performed if the MultiViews [option](#) is enabled. If *filename* resolves to a static text/html document (not a CGI script) and the Includes [option](#) is enabled, the file will be processed for server-side includes (see the [mod_include](#) documentation).

If the file specified by HeaderName contains the beginnings of an HTML document (<HTML>, <HEAD>, etc) then you will probably want to set [IndexOptions +SuppressHTMLPreamble](#), so that these tags are not repeated.

See also [ReadmeName](#).

IndexIgnore directive

Syntax: IndexIgnore *file* [*file*] ...

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

The IndexIgnore directive adds to the list of files to hide when listing a directory. *File* is a file extension, partial filename, wildcard expression or full filename for files to ignore. Multiple IndexIgnore directives add to the list, rather than replacing the list of ignored files. By default, the list contains `.`. Example:

```
IndexIgnore README .htaccess *~
```

IndexOptions directive

Syntax: IndexOptions *option* [*option*] ... (Apache 1.3.2 and earlier)

Syntax: IndexOptions [+|*option*] [|+|*option*] ... (Apache 1.3.3 and later)

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Base

Module: mod_autoindex

Compatibility: '+/-' syntax and merging of multiple IndexOptions directives is only available with Apache 1.3.3 and later; the FoldersFirst and DescriptionWidth options are only available with Apache 1.3.10 and later; the TrackModified option is only available with Apache 1.3.15 and later

The IndexOptions directive specifies the behavior of the directory indexing. *Option* can be one of

DescriptionWidth=[*n* | *] (*Apache 1.3.10 and later*)

The DescriptionWidth keyword allows you to specify the width of the description column in characters. If the keyword value is '*', then the column is automatically sized to the length of the longest filename in the display. **See the section on [AddDescription](#) for dangers inherent in truncating descriptions.**

FancyIndexing

This turns on fancy indexing of directories.

Note that in versions of Apache prior to 1.3.2, the FancyIndexing and IndexOptions directives will override each other. You should use IndexOptions FancyIndexing in preference to the standalone FancyIndexing directive. As of Apache 1.3.2, a standalone FancyIndexing directive is combined with any IndexOptions directive already specified for the current scope.

FoldersFirst (*Apache 1.3.10 and later*)

If this option is enabled, subdirectories in a FancyIndexed listing will *always* appear first, followed by normal files in the directory. The listing is basically broken into two components, the files and the subdirectories, and each is sorted separately and then displayed subdirectories-first. For instance, if the sort order is descending by name, and FoldersFirst is enabled, subdirectory Zed will be listed before subdirectory Beta, which will be listed before normal files Gamma and Alpha. **This option only has an effect if [FancyIndexing](#) is also enabled.**

IconHeight[=*pixels*] (*Apache 1.3 and later*)

Presence of this option, when used with **IconWidth**, will cause the server to include **HEIGHT** and **WIDTH** attributes in the **IMG** tag for the file icon. This allows browser to precalculate the page layout without having to wait until all the images have been loaded. If no value is given for the option, it defaults to the standard height of the icons supplied with the Apache software.

IconsAreLinks

This makes the icons part of the anchor for the filename, for fancy indexing.

IconWidth[=*pixels*] (*Apache 1.3 and later*)

Presence of this option, when used with **IconHeight**, will cause the server to include **HEIGHT** and **WIDTH** attributes in the **IMG** tag for the file icon. This allows browser to precalculate the page layout without having to wait until all the images have been loaded. If no value is given for the option, it defaults to the standard width of the icons supplied with the Apache software.

NameWidth=[*n* | *] (*Apache 1.3.2 and later*)

The **NameWidth** keyword allows you to specify the width of the filename column in bytes. If the keyword value is '*', then the column is automatically sized to the length of the longest filename in the display.

ScanHTMLTitles

This enables the extraction of the title from HTML documents for fancy indexing. If the file does not have a description given by [AddDescription](#) then httpd will read the document for the value of the **TITLE** tag. This is CPU and disk intensive.

SuppressColumnSorting

If specified, Apache will not make the column headings in a **FancyIndexed** directory listing into links for sorting. The default behavior is for them to be links; selecting the column heading will sort the directory listing by the values in that column. **Only available in Apache 1.3 and later.**

SuppressDescription

This will suppress the file description in fancy indexing listings. By default, no file descriptions are defined, and so the use of this option will regain 23 characters of screen space to use for something else. See [AddDescription](#) for information about setting the file description. See also the [DescriptionWidth](#) index option to limit the size of the description column.

SuppressHTMLPreamble (*Apache 1.3 and later*)

If the directory actually contains a file specified by the [HeaderName](#) directive, the module usually includes the contents of the file after a standard HTML preamble (<HTML>, <HEAD>, *et cetera*). The **SuppressHTMLPreamble** option disables this behavior, causing the module to start the display with the header file contents. The header file must contain appropriate HTML instructions in this case. If there is no header file, the preamble is generated as usual.

SuppressLastModified

This will suppress the display of the last modification date, in fancy indexing listings.

SuppressSize

This will suppress the file size in fancy indexing listings.

TrackModified (*Apache 1.3.15 and later*)

This returns the Last-Modified and ETag values for the listed directory in the HTTP header. It is only valid if the operating system and file system return legitimate `stat()` results. Most Unix systems do so, as do OS2's JFS and Win32's NTFS volumes. OS2 and Win32 FAT volumes, for example, do not. Once this feature is enabled, the client or proxy can track changes to the list of files when they perform a **HEAD** request. Note some operating systems correctly track new and removed files, but do not track changes for sizes or dates of the files within the directory.

There are some noticeable differences in the behavior of this directive in recent (post-1.3.0) versions of Apache.

Apache 1.3.2 and earlier:

The default is that no options are enabled. If multiple **IndexOptions** could apply to a directory, then the most specific one is taken complete; the options are not merged. For example:

```
<Directory /web/docs>
    IndexOptions FancyIndexing
</Directory>
<Directory /web/docs/spec>
```

```

    IndexOptions ScanHTMLTitles
  </Directory>

```

then only `ScanHTMLTitles` will be set for the `/web/docs/spec` directory.

Apache 1.3.3 and later:

Apache 1.3.3 introduced some significant changes in the handling of `IndexOptions` directives. In particular,

- Multiple `IndexOptions` directives for a single directory are now merged together. The result of the example above will now be the equivalent of `IndexOptions FancyIndexing ScanHTMLTitles`.
- The addition of the incremental syntax (*i.e.*, prefixing keywords with '+' or '-').

Whenever a '+' or '-' prefixed keyword is encountered, it is applied to the current `IndexOptions` settings (which may have been inherited from an upper-level directory). However, whenever an unprefixed keyword is processed, it clears all inherited options and any incremental settings encountered so far. Consider the following example:

```

IndexOptions +ScanHTMLTitles -IconsAreLinks FancyIndexing
IndexOptions +SuppressSize

```

The net effect is equivalent to `IndexOptions FancyIndexing +SuppressSize`, because the unprefixed `FancyIndexing` discarded the incremental keywords before it, but allowed them to start accumulating again afterward.

To unconditionally set the `IndexOptions` for a particular directory, clearing the inherited settings, specify keywords without either '+' or '-' prefixes.

IndexOrderDefault directive

Syntax: `IndexOrderDefault Ascending|Descending Name|Date|Size|Description`

Context: server config, virtual host, directory, `.htaccess`

Override: Indexes

Status: Base

Module: mod_autoindex

Compatibility: `IndexOrderDefault` is only available in Apache 1.3.4 and later.

The `IndexOrderDefault` directive is used in combination with the [FancyIndexing](#) index option. By default, fancyindexed directory listings are displayed in ascending order by filename; the `IndexOrderDefault` allows you to change this initial display order.

`IndexOrderDefault` takes two arguments. The first must be either `Ascending` or `Descending`, indicating the direction of the sort. The second argument must be one of the keywords `Name`, `Date`, `Size`, or `Description`, and identifies the primary key. The secondary key is *always* the ascending filename.

You can force a directory listing to only be displayed in a particular order by combining this directive with the [SuppressColumnSorting](#) index option; this will prevent the client from requesting the directory listing in a different order.

ReadmeName directive

Syntax: `ReadmeName filename`

Context: server config, virtual host, directory, `.htaccess`

Override: Indexes

Status: Base

Module: mod_autoindex

Compatibility: some features only available after 1.3.6; see text

The `ReadmeName` directive sets the name of the file that will be appended to the end of the index listing. *Filename* is the name of the file to include, and is taken to be relative to the location being indexed.

The *filename* argument is treated as a stub filename in Apache 1.3.6 and earlier, and as a relative URI in

later versions. Details of how it is handled may be found under the description of the [HeaderName](#) directive, which uses the same mechanism and changed at the same time as ReadmeName.

See also [HeaderName](#).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache module mod_cern_meta

This module provides for CERN httpd metafile semantics.

Status: Extension

Source File: mod_cern_meta.c

Module Identifier: cern_meta_module

Compatibility: Available in Apache 1.1 and later.

Summary

Emulate the CERN HTTPD Meta file semantics. Meta files are HTTP headers that can be output in addition to the normal range of headers for each file accessed. They appear rather like the Apache .asis files, and are able to provide a crude way of influencing the Expires: header, as well as providing other curiosities. There are many ways to manage meta information, this one was chosen because there is already a large number of CERN users who can exploit this module.

More information on the [CERN metafile semantics](#) is available.

Directives

- [MetaFiles](#)
- [MetaDir](#)
- [MetaSuffix](#)

MetaFiles directive

Syntax: MetaFiles on|off

Default: MetaFiles off

Context: per-directory config

Status: Base

Module: mod_cern_meta

Compatibility: MetaFiles is only available in Apache 1.3 and later.

Turns on/off Meta file processing on a per-directory basis. This option was introduced in Apache 1.3.

MetaDir directive

Syntax: MetaDir *directory*

Default: MetaDir .web

Context: (Apache prior to 1.3) server config

Context: (Apache 1.3) per-directory config

Module mod_cern_meta

Status: Base

Module: mod_cern_meta

Compatibility: MetaDir is only available in Apache 1.1 and later.

Specifies the name of the directory in which Apache can find meta information files. The directory is usually a 'hidden' subdirectory of the directory that contains the file being accessed. Set to "." to look in the same directory as the file.

MetaSuffix directive

Syntax: MetaSuffix *suffix*

Default: MetaSuffix .meta

Context: (Apache prior to 1.3) server config

Context: (Apache 1.3) per-directory config

Status: Base

Module: mod_cern_meta

Compatibility: MetaSuffix is only available in Apache 1.1 and later.

Specifies the file name suffix for the file containing the meta information. For example, the default values for the two directives will cause a request to DOCUMENT_ROOT/somedir/index.html to look in

DOCUMENT_ROOT/somedir/.web/index.html.meta and will use its contents to generate additional MIME header information.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_cgi

This module provides for execution of CGI scripts.

Status: Base

Source File: mod_cgi.c

Module Identifier: cgi_module

Summary

Any file that has the mime type `application/x-httpd-cgi` or handler `cgi-script` (Apache 1.1 or later) will be treated as a CGI script, and run by the server, with its output being returned to the client. Files acquire this type either by having a name containing an extension defined by the [AddType](#) directive, or by being in a [ScriptAlias](#) directory. Files that are not in a [ScriptAlias](#) directory, but which are of type `application/x-httpd-cgi` by virtue of an `AddType` directive, will still not be executed by the server unless `Options ExecCGI` is enabled. See the [Options](#) directive for more details.

When the server invokes a CGI script, it will add a variable called `DOCUMENT_ROOT` to the environment. This variable will contain the value of the [DocumentRoot](#) configuration variable.

For an introduction to using CGI scripts with Apache, see our tutorial on [Dynamic Content with CGI](#).

Directives

- [ScriptLog](#)
- [ScriptLogLength](#)
- [ScriptLogBuffer](#)

See also: [Options](#), [ScriptAlias](#), [AddType](#) and [AddHandler](#).

CGI Environment variables

The server will set the CGI environment variables as described in the [CGI specification](#), with the following provisions:

REMOTE_HOST

This will only be set if [HostnameLookups](#) is set to on (it is off by default), and if a reverse DNS lookup of the accessing host's address indeed finds a host name.

REMOTE_IDENT

This will only be set if [IdentityCheck](#) is set to on and the accessing host supports the ident protocol. Note that the contents of this variable cannot be relied upon because it can easily be faked, and if there is a proxy between the client and the server, it is usually totally useless.

REMOTE_USER

This will only be set if the CGI script is subject to authentication.

CGI Debugging

Debugging CGI scripts has traditionally been difficult, mainly because it has not been possible to study the output (standard output and error) for scripts which are failing to run properly. These directives, included in Apache 1.2 and later, provide more detailed logging of errors when they occur.

CGI Logfile Format

When configured, the CGI error log logs any CGI which does not execute properly. Each CGI script which fails to operate causes several lines of information to be logged. The first two lines are always of the format:

```
%% [time] request-line
%% HTTP-status CGI-script-filename
```

If the error is that CGI script cannot be run, the log file will contain an extra two lines:

```
%%error
error-message
```

Alternatively, if the error is the result of the script returning incorrect header information (often due to a bug in the script), the following information is logged:

```
%request
All HTTP request headers received
POST or PUT entity (if any)
%response
All headers output by the CGI script
%stdout
CGI standard output
%stderr
CGI standard error
```

(The %stdout and %stderr parts may be missing if the script did not output anything on standard output or standard error).

ScriptLog directive

Syntax: ScriptLog *filename*

Default: none

Context: server config

Status: mod_cgi

The `ScriptLog` directive sets the CGI script error logfile. If no `ScriptLog` is given, no error log is created. If given, any CGI errors are logged into the filename given as argument. If this is a relative file or path it is taken relative to the server root.

This log will be opened as the user the child processes run as, ie. the user specified in the main [User](#) directive. This means that either the directory the script log is in needs to be writable by that user or the file needs to be manually created and set to be writable by that user. If you place the script log in your main logs directory, do **NOT** change the directory permissions to make it writable by the user the child processes run as.

Note that script logging is meant to be a debugging feature when writing CGI scripts, and is not meant to be activated continuously on running servers. It is not optimized for speed or efficiency, and may have security problems if used in a manner other than that for which it was designed.

ScriptLogLength directive

Syntax: ScriptLogLength *bytes*

Default: 10385760

Context: server config

Status: mod_cgi

ScriptLogLength can be used to limit the size of the CGI script logfile. Since the logfile logs a lot of information per CGI error (all request headers, all script output) it can grow to be a big file. To prevent problems due to unbounded growth, this directive can be used to set an maximum file-size for the CGI logfile. If the file exceeds this size, no more information will be written to it.

ScriptLogBuffer

Syntax: ScriptLogBuffer *bytes*

Default: 1024

Context: server config

Status: mod_cgi

The size of any PUT or POST entity body that is logged to the file is limited, to prevent the log file growing too big too quickly if large bodies are being received. By default, up to 1024 bytes are logged, but this can be changed with this directive.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_digest

This module provides for user authentication using MD5 Digest Authentication.

Status: Extension

Source File: mod_digest.c

Module Identifier: digest_module

Compatibility: Available in Apache 1.1 and later.

Summary

This module implements an older version of the MD5 Digest Authentication specification which will probably not work with modern browsers. Please see [mod_auth_digest](#) for a module which implements the most recent version of the standard.

Directives

- [AuthDigestFile](#)

Using Digest Authentication

Using MD5 Digest authentication is very simple. Simply set up authentication normally. However, use "AuthType Digest" and "AuthDigestFile" instead of the normal "AuthType Basic" and "AuthUserFile". Everything else should remain the same.

MD5 authentication provides a more secure password system, but only works with supporting browsers. As of this writing (July 1996), the majority of browsers do not support digest authentication. Therefore, we do not recommend using this feature on a large Internet site. However, for personal and intra-net use, where browser users can be controlled, it is ideal.

AuthDigestFile directive

Syntax: AuthDigestFile *filename*

Context: directory, .htaccess

Override: AuthConfig

Status: Base

Module: mod_digest

The AuthDigestFile directive sets the name of a textual file containing the list of users and encoded passwords for digest authentication. *Filename* is the absolute path to the user file.

The digest file uses a special format. Files in this format can be created using the "htdigest" utility found in the support/ subdirectory of the Apache distribution.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_dir

This module provides for "trailing slash" redirects and serving directory index files.

Status: Base

Source File: mod_dir.c

Module Identifier: dir_module

Summary

The index of a directory can come from one of two sources:

- A file written by the user, typically called `index.html`. The [DirectoryIndex](#) directive sets the name of this file. This is controlled by `mod_dir`.
- Otherwise, a listing generated by the server. This is provided by [mod_autoindex](#).

The two functions are separated so that you can completely remove (or replace) automatic index generation should you want to.

A "trailing slash" redirect is issued when the server receives a request for a URL `http://servername/foo/dirname` where `dirname` is a directory. Directories require a trailing slash, so `mod_dir` issues a redirect to `http://servername/foo/dirname/`.

Directives

- [DirectoryIndex](#)

DirectoryIndex directive

Syntax: `DirectoryIndex local-url [local-url] ...`

Default: `DirectoryIndex index.html`

Context: server config, virtual host, directory, `.htaccess`

Override: Indexes

Status: Base

Module: `mod_dir`

The `DirectoryIndex` directive sets the list of resources to look for, when the client requests an index of the directory by specifying a / at the end of the a directory name. *Local-url* is the (%-encoded) URL of a document on the server relative to the requested directory; it is usually the name of a file in the directory. Several URLs may be given, in which case the server will return the first one that it finds. If none of the resources exist and the `Indexes` option is set, the server will generate its own listing of the directory.

Example:

```
DirectoryIndex index.html
```

then a request for `http://myserver/docs/` would return `http://myserver/docs/index.html` if it exists, or would list the directory if it did not.

Note that the documents do not need to be relative to the directory;

```
DirectoryIndex index.html index.txt /cgi-bin/index.pl
```

would cause the CGI script /cgi-bin/index.pl to be executed if neither index.html or index.txt existed in a directory.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache module mod_env

This module provides for modifying the environment which is passed to CGI scripts and SSI pages.

Status: Base

Source File: mod_env.c

Module Identifier: env_module

Compatibility: Available in Apache 1.1 and later.

Summary

This module allows for control of the environment that will be provided to CGI scripts and SSI pages. Environment variables may be passed from the shell which invoked the httpd process. Alternatively, environment variables may be set or unset within the configuration process.

For additional information, we provide a document on [Environment Variables in Apache](#).

Directives

- [PassEnv](#)
- [SetEnv](#)
- [UnsetEnv](#)

PassEnv directive

Syntax: PassEnv *env-variable* [*env-variable*] ...

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_env

Compatibility: PassEnv is only available in Apache 1.1 and later. Directory and .htaccess context is available in Apache 1.3.7 and later.

Specifies one or more environment variables to pass to CGI scripts and SSI pages from the environment of the shell which invoked the httpd process. Example:

```
PassEnv LD_LIBRARY_PATH
```

SetEnv directive

Syntax: SetEnv *env-variable value*

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_env

Compatibility: SetEnv is only available in Apache 1.1 and later. Directory and .htaccess context is available in Apache 1.3.7 and later.

Sets an environment variable, which is then passed on to CGI scripts and SSI pages. Example:

```
SetEnv SPECIAL_PATH /foo/bin
```

UnsetEnv directive

Syntax: UnsetEnv *env-variable [env-variable] ...*

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_env

Compatibility: UnsetEnv is only available in Apache 1.1 and later. Directory and .htaccess context is available in Apache 1.3.7 and later.

Removes one or more environment variables from those passed on to CGI scripts and SSI pages. Example:

```
UnsetEnv LD_LIBRARY_PATH
```

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_example

This module illustrates many of the aspects of the [Apache 1.2 API](#) and, when used, demonstrates the manner in which module callbacks are triggered by the server.

Status: Extension

Source File: mod_example.c

Module Identifier: example_module

Compatibility: Available in Apache 1.2 and later.

Summary

The files in the `src/modules/example` directory under the Apache distribution directory tree are provided as an example to those that wish to write modules that use the Apache API.

The main file is `mod_example.c`, which illustrates all the different callback mechanisms and call syntaxes. By no means does an add-on module need to include routines for all of the callbacks - quite the contrary!

The example module is an actual working module. If you link it into your server, enable the "example-handler" handler for a location, and then browse to that location, you will see a display of some of the tracing the example module did as the various callbacks were made.

Directives

- [Example](#)

Compiling the example module

To include the example module in your server, follow the steps below:

1. Uncomment the "AddModule modules/example/mod_example" line near the bottom of the `src/Configuration` file. If there isn't one, add it; it should look like this:

```
AddModule modules/example/mod_example.o
```

2. Run the `src/Configure` script ("cd src; ./Configure"). This will build the Makefile for the server itself, and update the `src/modules/Makefile` for any additional modules you have requested from beneath that subdirectory.
3. Make the server (run "make" in the `src` directory).

To add another module of your own:

- A. `mkdir src/modules/mymodule`
- B. `cp src/modules/example/* src/modules/mymodule`
- C. Modify the files in the new directory.
- D. Follow steps [1] through [3] above, with appropriate changes.

Using the mod_example Module

To activate the example module, include a block similar to the following in your srm.conf file:

```
<Location /example-info>
    SetHandler example-handler
</Location>
```

As an alternative, you can put the following into a [.htaccess](#) file and then request the file "test.example" from that location:

```
AddHandler example-handler .example
```

After reloading/restarting your server, you should be able to browse to this location and see the brief display mentioned earlier.

Example directive

Syntax: Example

Default: None

Context: server config, virtual host, directory, .htaccess

Override: Options

Status: Extension

Module: mod_example

Compatibility: Example is only available in Apache 1.2 and later.

The Example directive just sets a demonstration flag which the example module's content handler displays. It takes no arguments. If you browse to an URL to which the example content-handler applies, you will get a display of the routines within the module and how and in what order they were called to service the document request. The effect of this directive one can observe under the point "Example directive declared here: YES/NO".

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_expires

This module provides for the generation of Expires HTTP headers according to user-specified criteria.

Status: Extension

Source File: mod_expires.c

Module Identifier: expires_module

Compatibility: Available in Apache 1.2 and later.

Summary

This module controls the setting of the Expires HTTP header in server responses. The expiration date can set to be relative to either the time the source file was last modified, or to the time of the client access.

The Expires HTTP header is an instruction to the client about the document's validity and persistence. If cached, the document may be fetched from the cache rather than from the source until this time has passed. After that, the cache copy is considered "expired" and invalid, and a new copy must be obtained from the source.

Directives

- [ExpiresActive](#)
- [ExpiresByType](#)
- [ExpiresDefault](#)

Alternate Interval Syntax

The [ExpiresDefault](#) and [ExpiresByType](#) directives can also be defined in a more readable syntax of the form:

```
ExpiresDefault "<base> [plus] {<num> <type>}"*"  
ExpiresByType type/encoding "<base> [plus] {<num> <type>}"*
```

where <base> is one of:

- access
- now (equivalent to 'access')
- modification

The 'plus' keyword is optional. <num> should be an integer value [acceptable to atoi()], and <type> is one of:

- years
- months
- weeks
- days
- hours
- minutes

- seconds

For example, any of the following directives can be used to make documents expire 1 month after being accessed, by default:

```
ExpiresDefault "access plus 1 month"
ExpiresDefault "access plus 4 weeks"
ExpiresDefault "access plus 30 days"
```

The expiry time can be fine-tuned by adding several '<num> <type>' clauses:

```
ExpiresByType text/html "access plus 1 month 15 days 2 hours"
ExpiresByType image/gif "modification plus 5 hours 3 minutes"
```

Note that if you use a modification date based setting, the Expires header will **not** be added to content that does not come from a file on disk. This is due to the fact that there is no modification time for such content.

ExpiresActive directive

Syntax: ExpiresActive on|off

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Extension

Module: mod_expires

This directive enables or disables the generation of the Expires header for the document realm in question. (That is, if found in an .htaccess file, for instance, it applies only to documents generated from that directory.) If set to *Off*, no Expires header will be generated for any document in the realm (unless overridden at a lower level, such as an .htaccess file overriding a server config file). If set to *On*, the header will be added to served documents according to the criteria defined by the [ExpiresByType](#) and [ExpiresDefault](#) directives (*q.v.*).

Note that this directive does not guarantee that an Expires header will be generated. If the criteria aren't met, no header will be sent, and the effect will be as though this directive wasn't even specified.

ExpiresByType directive

Syntax: ExpiresByType *mime-type* <code>seconds

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Extension

Module: mod_expires

This directive defines the value of the Expires header generated for documents of the specified type (*e.g.*, `text/html`). The second argument sets the number of seconds that will be added to a base time to construct the expiration date.

The base time is either the last modification time of the file, or the time of the client's access to the document. Which should be used is specified by the <code> field; **M** means that the file's last modification time should be used as the base time, and **A** means the client's access time should be used.

The difference in effect is subtle. If *M* is used, all current copies of the document in all caches will expire at the same time, which can be good for something like a weekly notice that's always found at the same URL. If *A* is used, the date of expiration is different for each client; this can be good for image files that don't change very often, particularly for a set of related documents that all refer to the same images (*i.e.*, the images will be accessed repeatedly within a relatively short timespan).

Example:

```
ExpiresActive On # enable expirations
```

Apache module mod_expires

```
ExpiresByType image/gif A2592000 # expire GIF images after a month
                                     # in the client's cache
ExpiresByType text/html M604800  # HTML documents are good for a
                                     # week from the time they were
                                     # changed, period
```

Note that this directive only has effect if `ExpiresActive On` has been specified. It overrides, for the specified MIME type *only*, any expiration date set by the [ExpiresDefault](#) directive.

You can also specify the expiration time calculation using an [alternate syntax](#), described later in this document.

ExpiresDefault directive

Syntax: ExpiresDefault *<code>seconds*

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Status: Extension

Module: mod_expires

This directive sets the default algorithm for calculating the expiration time for all documents in the affected realm. It can be overridden on a type-by-type basis by the [ExpiresByType](#) directive. See the description of that directive for details about the syntax of the argument, and the [alternate syntax](#) description as well.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_headers

This module provides for the customization of HTTP response headers.

Status: Extension

Source File: mod_headers.c

Module Identifier: headers_module

Compatibility: Available in Apache 1.2 and later.

Summary

This module provides a directive to control the sending of HTTP headers. Headers can be merged, replaced or removed.

Directives

- [Header](#)
-

Header directive

Syntax: Header set|append|add *header value*

Syntax: Header unset *header*

Context: server config, virtual host, access.conf, .htaccess

Override: FileInfo

Status: Extension

Module: mod_header

This directive can replace, merge or remove HTTP response headers. The action it performs is determined by the first argument. This can be one of the following values:

- **set**
The response header is set, replacing any previous header with this name
- **append**
The response header is appended to any existing header of the same name. When a new value is merged onto an existing header it is separated from the existing header with a comma. This is the HTTP standard way of giving a header multiple values.
- **add**
The response header is added to the existing set of headers, even if this header already exists. This can result in two (or more) headers having the same name. This can lead to unforeseen consequences, and in general "append" should be used instead.
- **unset**
The response header of this name is removed, if it exists. If there are multiple headers of the same name, all will be removed.

This argument is followed by a header name, which can include the final colon, but it is not required. Case is ignored. For add, append and set a value is given as the third argument. If this value contains spaces, it should be surrounded by double quotes. For

unset, no value should be given.

Order of Processing

The Header directive can occur almost anywhere within the server configuration. It is valid in the main server config and virtual host sections, inside <Directory>, <Location> and <Files> sections, and within .htaccess files.

The Header directives are processed in the following order:

1. main server
2. virtual host
3. <Directory> sections and .htaccess
4. <Location>
5. <Files>

Order is important. These two headers have a different effect if reversed:

```
Header append Author "John P. Doe"  
Header unset Author
```

This way round, the Author header is not set. If reversed, the Author header is set to "John P. Doe".

The Header directives are processed just before the response is sent by its handler. These means that some headers that are added just before the response is sent cannot be unset or overridden. This includes headers such as "Date" and "Server".

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_imap

This module provides for server-side imagemap processing.

Status: Base

Source File: mod_imap.c

Module Identifier: imap_module

Compatibility: Available in Apache 1.1 and later.

Summary

This module processes `.map` files, thereby replacing the functionality of the `imagemap` CGI program. Any directory or document type configured to use the handler `imap-file` (using either [AddHandler](#) or [SetHandler](#)) will be processed by this module.

The following directive will activate files ending with `.map` as imagemap files:

```
AddHandler imap-file map
```

Note that the following is still supported:

```
AddType application/x-httpd-imap map
```

However, we are trying to phase out "magic MIME types" so we are deprecating this method.

Directives

- [ImapMenu](#)
- [ImapDefault](#)
- [ImapBase](#)

New Features

The `imagemap` module adds some new features that were not possible with previously distributed `imagemap` programs.

- URL references relative to the Referer: information.
- Default `<BASE>` assignment through a new `map` directive `base`.
- No need for `imagemap.conf` file.
- Point references.
- Configurable generation of `imagemap` menus.

Imagemap File

The lines in the `imagemap` files can have one of several formats:

```
directive value [x,y ...]  
directive value "Menu text" [x,y ...]
```

```
directive value x,y ... "Menu text"
```

The directive is one of `base`, `default`, `poly`, `circle`, `rect`, or `point`. The value is an absolute or relative URL, or one of the special values listed below. The coordinates are `x, y` pairs separated by whitespace. The quoted text is used as the text of the link if a imagemap menu is generated. Lines beginning with `#` are comments.

Imagemap File Directives

There are six directives allowed in the imagemap file. The directives can come in any order, but are processed in the order they are found in the imagemap file.

base Directive

Has the effect of `<BASE HREF="value">`. The non-absolute URLs of the map-file are taken relative to this value. The `base` directive overrides `ImapBase` as set in a `.htaccess` file or in the server configuration files. In the absence of an `ImapBase` configuration directive, `base` defaults to `http://server_name/`.

`base_uri` is synonymous with `base`. Note that a trailing slash on the URL is significant.

default Directive

The action taken if the coordinates given do not fit any of the `poly`, `circle` or `rect` directives, and there are no `point` directives. Defaults to `nocontent` in the absence of an `ImapDefault` configuration setting, causing a status code of 204 No Content to be returned. The client should keep the same page displayed.

poly Directive

Takes three to one-hundred points, and is obeyed if the user selected coordinates fall within the polygon defined by these points.

circle

Takes the center coordinates of a circle and a point on the circle. Is obeyed if the user selected point is with the circle.

rect Directive

Takes the coordinates of two opposing corners of a rectangle. Obeyed if the point selected is within this rectangle.

point Directive

Takes a single point. The point directive closest to the user selected point is obeyed if no other directives are satisfied. Note that `default` will not be followed if a `point` directive is present and valid coordinates are given.

Values

The values for each of the directives can any of the following:

a URL

The URL can be relative or absolute URL. Relative URLs can contain `'..'` syntax and will be resolved relative to the `base` value.

`base` itself will not resolved according to the current value. A statement `base mailto:` will work properly, though.

map

Equivalent to the URL of the imagemap file itself. No coordinates are sent with this, so a menu will be generated unless `ImapMenu` is set to `'none'`.

menu

Synonymous with `map`.

referer

Equivalent to the URL of the referring document. Defaults to `http://servername/` if no `Referer:` header was present.

nocontent

Sends a status code of 204 No Content, telling the client to keep the same page displayed. Valid for all but `base`.

error

Fails with a 500 Server Error. Valid for all but `base`, but sort of silly for anything but `default`.

Coordinates

0,0 200,200

A coordinate consists of an *x* and a *y* value separated by a comma. The coordinates are separated from each other by whitespace. To accommodate the way Lynx handles imagemaps, should a user select the coordinate 0,0, it is as if no coordinate had been selected.

Quoted Text

"Menu Text"

After the value or after the coordinates, the line optionally may contain text within double quotes. This string is used as the text for the link if a menu is generated:

```
<a HREF="http://foo.com/">Menu text</a>
```

If no quoted text is present, the name of the link will be used as the text:

```
<a HREF="http://foo.com/">http://foo.com</a>
```

It is impossible to escape double quotes within this text.

Example Mapfile

```
#Comments are printed in a 'formatted' or 'semiformatted' menu.
#And can contain html tags. <hr>
base referer
poly map "Could I have a menu, please?" 0,0 0,10 10,10 10,0
rect .. 0,0 77,27 "the directory of the referer"
circle http://www.inetnebr.com/lincoln/feedback/ 195,0 305,27
rect another_file "in same directory as referer" 306,0 419,27
point http://www.zyzyzyva.com/ 100,100
point http://www.tripod.com/ 200,200
rect mailto:nate@tripod.com 100,150 200,0 "Bugs?"
```

Referencing your mapfile

```
<A HREF="/maps/imagemap1.map">
<IMG ISMAP SRC="/images/imagemap1.gif">
</A>
```

ImapMenu directive

Syntax: ImapMenu none|formatted|semiformatted|unformatted

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Module: mod_imap

Compatibility: ImapMenu is only available in Apache 1.1 and later.

The ImapMenu directive determines the action taken if an imagemap file is called without valid coordinates.

none

If ImapMenu is none, no menu is generated, and the default action is performed.

formatted

A formatted menu is the simplest menu. Comments in the imagemap file are ignored. A level one header is printed, then an hrule, then the links each on a separate line. The menu has a consistent, plain look close to that of a directory listing.

semiformatted

In the semiformatted menu, comments are printed where they occur in the imagemap file. Blank lines are turned into

HTML breaks. No header or hrule is printed, but otherwise the menu is the same as a formatted menu.

unformatted

Comments are printed, blank lines are ignored. Nothing is printed that does not appear in the imagemap file. All breaks and headers must be included as comments in the imagemap file. This gives you the most flexibility over the appearance of your menus, but requires you to treat your map files as HTML instead of plaintext.

ImapDefault directive

Syntax: ImapDefault error|nocontent|map|referer|*URL*

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Module: mod_imap

Compatibility: ImapDefault is only available in Apache 1.1 and later.

The ImapDefault directive sets the default `default` used in the imagemap files. Its value is overridden by a `default` directive within the imagemap file. If not present, the `default` action is `nocontent`, which means that a 204 No Content is sent to the client. In this case, the client should continue to display the original page.

ImapBase directive

Syntax: ImapBase map|referer|*URL*

Context: server config, virtual host, directory, .htaccess

Override: Indexes

Module: mod_imap

Compatibility: ImapBase is only available in Apache 1.1 and later.

The ImapBase directive sets the default `base` used in the imagemap files. Its value is overridden by a `base` directive within the imagemap file. If not present, the `base` defaults to `http://servername/`.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_include

This module provides for documents with Server Side Includes (SSI).

Status: Base

Source File: mod_include.c

Module Identifier: includes_module

Summary

This module provides a handler which will process files before they are sent to the client. The processing is controlled by specially formatted SGML comments, referred to as *elements*. These elements allow conditional text, the inclusion other files or programs, as well as the setting and printing of environment variables.

For an introduction to this topic, we also provide a [tutorial on Server Side Includes](#).

Directives

- [XBitHack](#)

See also: [Options](#) and [AddHandler](#).

Enabling Server-Side Includes

Any document with handler of "server-parsed" will be parsed by this module, if the `Includes` option is set. If documents containing server-side include directives are given the extension `.shtml`, the following directives will make Apache parse them and assign the resulting document the mime type of `text/html`:

```
AddType text/html .shtml
AddHandler server-parsed .shtml
```

The following directive must be given for the directories containing the `shtml` files (typically in a `<Directory>` section, but this directive is also valid `.htaccess` files if `AllowOverride Options` is set):

```
Options +Includes
```

Alternatively the [XBitHack](#) directive can be used to parse normal (`text/html`) files, based on file permissions.

For backwards compatibility, documents with mime type `text/x-server-parsed-html` or `text/x-server-parsed-html3` will also be parsed (and the resulting output given the mime type `text/html`).

Basic Elements

The document is parsed as an HTML document, with special commands embedded as SGML comments. A command has the syntax:

```
<!--#element attribute=value attribute=value ... -->
```

The value will often be enclosed in double quotes; many commands only allow a single attribute-value pair. Note that the comment terminator (--) should be preceded by whitespace to ensure that it isn't considered part of an SSI token.

The allowed elements are:

config

This command controls various aspects of the parsing. The valid attributes are:

`errmsg`

The value is a message that is sent back to the client if an error occurs whilst parsing the document.

`sizefmt`

The value sets the format to be used which displaying the size of a file. Valid values are `bytes` for a count in bytes, or `abbrev` for a count in Kb or Mb as appropriate.

`timefmt`

The value is a string to be used by the `strftime(3)` library routine when printing dates.

echo

This command prints one of the include variables, defined below. If the variable is unset, it is printed as `(none)`. Any dates printed are subject to the currently configured `timefmt`. Attributes:

`var`

The value is the name of the variable to print.

`encoding`

Specifies how Apache should encode special characters contained in the variable before outputting them. If set to "none", no encoding will be done. If set to "url", then URL encoding (also known as %-encoding; this is appropriate for use within URLs in links, etc.) will be performed. At the start of an `echo` element, the default is set to "entity", resulting in entity encoding (which is appropriate in the context of a block-level HTML element, eg. a paragraph of text). This can be changed by adding an `encoding` attribute, which will remain in effect until the next `encoding` attribute is encountered or the element ends, whichever comes first. Note that the `encoding` attribute must *precede* the corresponding `var` attribute to be effective, and that only special characters as defined in the ISO-8859-1 character encoding will be encoded. This encoding process may not have the desired result if a different character encoding is in use. Apache 1.3.12 and above; previous versions do no encoding.

exec

The `exec` command executes a given shell command or CGI script. The [IncludesNOEXEC Option](#) disables this command completely. The valid attributes are:

`cgi`

The value specifies a (%-encoded) URL relative path to the CGI script. If the path does not begin with a (/), then it is taken to be relative to the current document. The document referenced by this path is invoked as a CGI script, even if the server would not normally recognize it as such. However, the directory containing the script must be enabled for CGI scripts (with [ScriptAlias](#) or the [ExecCGI Option](#)).

The CGI script is given the `PATH_INFO` and query string (`QUERY_STRING`) of the original request from the client; these cannot be specified in the URL path. The include variables will be available to the script in addition to the standard [CGI](#) environment.

If the script returns a `Location:` header instead of output, then this will be translated into an HTML anchor.

The `include virtual` element should be used in preference to `exec cgi`.

`cmd`

The server will execute the given string using `/bin/sh`. The include variables are available to the command.

fsize

This command prints the size of the specified file, subject to the `sizefmt` format specification. Attributes:

`file`

The value is a path relative to the directory containing the current document being parsed.

`virtual`

The value is a (%-encoded) URL-path relative to the current document being parsed. If it does not begin with a slash (/) then it is taken to be relative to the current document.

flastmod

This command prints the last modification date of the specified file, subject to the `timefmt` format specification. The attributes are the same as for the `fsize` command.

include

This command inserts the text of another document or file into the parsed file. Any included file is subject to the usual access control. If the directory containing the parsed file has the [Option](#) `IncludesNOEXEC` set, and the including the document would cause a program to be executed, then it will not be included; this prevents the execution of CGI scripts. Otherwise CGI scripts are invoked as normal using the complete URL given in the command, including any query string.

An attribute defines the location of the document; the inclusion is done for each attribute given to the `include` command. The valid attributes are:

`file`

The value is a path relative to the directory containing the current document being parsed. It cannot contain `./`, nor can it be an absolute path. The `virtual` attribute should always be used in preference to this one.

`virtual`

The value is a (%-encoded) URL relative to the current document being parsed. The URL cannot contain a scheme or hostname, only a path and an optional query string. If it does not begin with a slash (/) then it is taken to be relative to the current document.

A URL is constructed from the attribute, and the output the server would return if the URL were accessed by the client is included in the parsed output. Thus included files can be nested.

printenv

This prints out a listing of all existing variables and their values. Starting with Apache 1.3.12, special characters are entity encoded (see the [echo](#) element for details) before being output. No attributes.

For example: `<!--#printenv -->`

Apache 1.2 and above.

set

This sets the value of a variable. Attributes:

`var`

The name of the variable to set.

`value`

The value to give a variable.

For example: `<!--#set var="category" value="help" -->`

Apache 1.2 and above.

Include Variables

In addition to the variables in the standard CGI environment, these are available for the `echo` command, for `if` and `elif`, and to any program invoked by the document.

`DATE_GMT`

The current date in Greenwich Mean Time.

`DATE_LOCAL`

Apache module `mod_include`

The current date in the local time zone.

`DOCUMENT_NAME`

The filename (excluding directories) of the document requested by the user.

`DOCUMENT_URI`

The (%-decoded) URL path of the document requested by the user. Note that in the case of nested include files, this is *not* then URL for the current document.

`LAST_MODIFIED`

The last modification date of the document requested by the user.

Variable Substitution

Variable substitution is done within quoted strings in most cases where they may reasonably occur as an argument to an SSI directive. This includes the `config`, `exec`, `flastmod`, `fsize`, `include`, and `set` directives, as well as the arguments to conditional operators. You can insert a literal dollar sign into the string using backslash quoting:

```
<!--#if expr="$a = \$test" -->
```

If a variable reference needs to be substituted in the middle of a character sequence that might otherwise be considered a valid identifier in its own right, it can be disambiguated by enclosing the reference in braces, *à la* shell substitution:

```
<!--#set var="Zed" value="\${REMOTE_HOST}_\${REQUEST_METHOD}" -->
```

This will result in the `Zed` variable being set to `"X_Y"` if `REMOTE_HOST` is `"X"` and `REQUEST_METHOD` is `"Y"`.

EXAMPLE: the below example will print `"in foo"` if the `DOCUMENT_URI` is `/foo/file.html`, `"in bar"` if it is `/bar/file.html` and `"in neither"` otherwise:

```
<!--#if expr="\$DOCUMENT_URI\" = \"/foo/file.html\" -->
in foo
<!--#elif expr="\$DOCUMENT_URI\" = \"/bar/file.html\" -->
in bar
<!--#else -->
in neither
<!--#endif -->
```

Flow Control Elements

These are available in Apache 1.2 and above. The basic flow control elements are:

```
<!--#if expr="test_condition" -->
<!--#elif expr="test_condition" -->
<!--#else -->
<!--#endif -->
```

The **if** element works like an if statement in a programming language. The test condition is evaluated and if the result is true, then the text until the next **elif**, **else**, or **endif** element is included in the output stream.

The **elif** or **else** statements are be used the put text into the output stream if the original test_condition was false. These elements are optional.

The **endif** element ends the **if** element and is required.

test_condition is one of the following:

string

true if *string* is not empty

```
string1 = string2
string1 != string2
string1 < string2
string1 <= string2
string1 > string2
string1 >= string2
```

Compare string1 with string 2. If string2 has the form */string/* then it is compared as a regular expression. Regular expressions have the same syntax as those found in the Unix egrep command.

(*test_condition*)

true if *test_condition* is true

! *test_condition*

true if *test_condition* is false

test_condition1 && *test_condition2*

true if both *test_condition1* and *test_condition2* are true

test_condition1 || *test_condition2*

true if either *test_condition1* or *test_condition2* is true

"=" and "!=" bind more tightly than "&&" and "||". "!" binds most tightly. Thus, the following are equivalent:

```
<!--#if expr="$a = test1 && $b = test2" -->
<!--#if expr="($a = test1) && ($b = test2)" -->
```

Anything that's not recognized as a variable or an operator is treated as a string. Strings can also be quoted: *'string'*. Unquoted strings can't contain whitespace (blanks and tabs) because it is used to separate tokens such as variables. If multiple strings are found in a row, they are concatenated using blanks. So,

```
string1 string2 results in string1 string2
'string1 string2' results in string1 string2
```

Using Server Side Includes for ErrorDocuments

There is [a document](#) which describes how to use the features of mod_include to offer internationalized customized server error documents.

XBitHack directive

Syntax: XBitHack on|off|full

Default: XBitHack off

Context: server config, virtual host, directory, .htaccess

Override: Options

Status: Base

Module: mod_include

The XBitHack directives controls the parsing of ordinary html documents. This directive only affects files associated with the MIME type `text/html`. XBitHack can take on the following values:

off

No special treatment of executable files.

on

Any file that has the user-execute bit set will be treated as a server-parsed html document.

full

As for on but also test the group-execute bit. If it is set, then set the Last-modified date of the returned file to be the last modified time of the file. If it is not set, then no last-modified date is sent. Setting this bit allows clients and proxies to cache the result of the request.

Note: you would not want to use this, for example, when you #include a CGI that produces different output on each hit (or potentially depends on the hit).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_info

This module provides a comprehensive overview of the server configuration including all installed modules and directives in the configuration files.

Status: Extension

Source File: mod_info.c

Module Identifier: info_module

Compatibility: Available in Apache 1.1 and later.

Directives

- [AddModuleInfo](#)

Using mod_info

To configure it, add the following to your `access.conf` file.

```
<Location /server-info>
SetHandler server-info
</Location>
```

You may wish to add a [<Limit>](#) clause inside the [location](#) directive to limit access to your server configuration information.

Once configured, the server information is obtained by accessing `http://your.host.dom/server-info`

Note that the configuration files are read by the module at run-time, and therefore the display may *not* reflect the running server's active configuration if the files have been changed since the server was last reloaded. Also, the configuration files must be readable by the user as which the server is running (see the [User](#) directive), or else the directive settings will not be listed.

It should also be noted that if mod_info is compiled into the server, its handler capability is available in *all* configuration files, including *per-directory* files (e.g., `.htaccess`). This may have security-related ramifications for your site.

AddModuleInfo

Syntax: `AddModuleInfo module-name string`

Context: server config, virtual host

Status: Extension

Module: mod_info

Compatibility: Apache 1.3 and above

This allows the content of *string* to be shown as HTML interpreted, **Additional Information** for the module *module-name*. Example:

```
AddModuleInfo mod_auth.c 'See <A
  HREF="http://www.apache.org/docs/mod/mod_auth.html">http://www.apache.org/docs/mod/mod_auth.html</A>'
```

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_isapi

This module supports ISAPI Extensions within Apache for Windows.

Status: Base

Source File: mod_isapi.c

Module Identifier: isapi_module

Compatibility: WIN32 only

Summary

This module implements the Internet Server extension API. It allows Internet Server extensions (*e.g.* ISAPI .dll modules) to be served by Apache for Windows, subject to the noted restrictions.

ISAPI extension modules (.dll files) are written by third parties. The Apache Group does not author these modules, so we provide no support for them. Please contact the ISAPI's author directly if you are experiencing problems running their ISAPI extension.

Please do not post such problems to Apache's lists or bug reporting pages.

Directives

- [ISAPIReadAheadBuffer](#)
- [ISAPILogNotSupported](#)
- [ISAPIAppendLogToErrors](#)
- [ISAPIAppendLogToQuery](#)

Usage

In the server configuration file, use the `AddHandler` directive to associate ISAPI files with the `isapi-isa` handler, and map it to the with their file extensions. To enable any .dll file to be processed as an ISAPI extension, edit the `httpd.conf` file and add the following line:

```
AddHandler isapi-isa .dll
```

ISAPI extensions are governed by the same permissions and restrictions as CGI scripts. That is, `Options ExecCGI` must be set for the directory that contains the ISAPI .dll file.

Review the [Additional Notes](#) and the [Programmer's Journal](#) for additional details and clarification of the specific ISAPI support offered by mod_isapi.

Additional Notes

Apache's ISAPI implementation conforms to all of the ISAPI 2.0 specification, except for the "Microsoft-specific" extensions dealing with asynchronous I/O. Apache's I/O model does not allow asynchronous reading and writing in a manner that the ISAPI could access. If an ISA tries to access unsupported features, including async I/O, a message is placed in the error log to help with debugging. Since these messages can become a flood, a new directive; `ISAPILogNotSupported Off`, is introduced in Apache 1.3.13.

Some servers, like Microsoft IIS, load the ISA into the server, and keep it loaded until memory usage is too high, or specific configuration options are used. Apache currently loads and unloads the ISA for each request. This is inefficient, but Apache's request model makes this method the only method that currently works. Apache 2.0 is expected to support more effective loading and caching methods, with more precise control over individual ISAPI modules and directories.

Also, remember that while Apache supports ISAPI Extensions, it **does not support ISAPI Filters**. Support for filters may be added at a later date, but no support is planned at this time.

Programmer's Journal

If you are programming Apache 1.3 mod_isapi modules, you must limit your calls to `ServerSupportFunction` to the following directives:

HSE_REQ_SEND_URL_REDIRECT_RESP

Redirect the user to another location.
This must be a fully qualified URL (e.g. `http://server/location`).

HSE_REQ_SEND_URL

Redirect the user to another location.
This cannot be a fully qualified URL, you are not allowed to pass the protocol or a server name (e.g. `simply /location`).
This redirection is handled by the server, not the browser.

Warning: in their recent documentation, Microsoft appears to have abandoned the distinction between the two `HSE_REQ_SEND_URL` functions. Apache continues to treat them as two distinct functions with different requirements and behaviors.

HSE_REQ_SEND_RESPONSE_HEADER

Apache accepts a response body following the header if it follows the blank line (two consecutive newlines) in the headers string argument. This body cannot contain NULLs, since the headers argument is NULL terminated.

HSE_REQ_DONE_WITH_SESSION

Apache considers this a no-op, since the session will be finished when the ISAPI returns from processing.

HSE_REQ_MAP_URL_TO_PATH

Apache will translate a virtual name to a physical name.

HSE_APPEND_LOG_PARAMETER *Apache 1.3.13 and later*

This logged message may be captured in any of the following logs:

- in the `"%{isapi-parameter}n"` component in a `CustomLog` directive
- in the `%q` log component with the `ISAPIAppendLogToQuery On` directive
- in the error log with the `ISAPIAppendLogToErrors On` directive

The first option, the `%{isapi-parameter}n` component, is always available and preferred.

HSE_REQ_IS_KEEP_CONN *Apache 1.3.13 and later*

Will return the negotiated Keep-Alive status.

HSE_REQ_SEND_RESPONSE_HEADER_EX *Apache 1.3.13 and later*

Will behave as documented, although the `fKeepConn` flag is ignored.

HSE_REQ_IS_CONNECTED *Apache 1.3.13 and later*

Will report false if the request has been aborted.

Apache returns `FALSE` to any unsupported call to `ServerSupportFunction`, and sets the `GetLastError` value to

ERROR_INVALID_PARAMETER.

Prior to Apache 1.3.13, ReadClient was a noop, and any request with a request body greater than 48kb was rejected by mod_isapi. As of Apache 1.3.13, ReadClient now retrieves the request body exceeding the initial buffer (defined by ISAPIReadAheadBuffer). Based on the ISAPIReadAheadBuffer setting (number of bytes to buffer prior to calling the ISAPI handler) shorter requests are sent complete to the extension when it is invoked. If the request is longer, the ISAPI extension must use ReadClient to retrieve the remaining request body.

WriteClient is supported, but only with the HSE_IO_SYNC flag or no option flag (value of 0). Any other WriteClient request will be rejected with a return value of FALSE, and a GetLastError value of ERROR_INVALID_PARAMETER.

GetServerVariable is supported, although extended server variables do not exist (as defined by other servers.) All the usual Apache CGI environment variables are available from GetServerVariable. As of Apache 1.3.13, the ALL_HTTP and ALL_RAW and variables are now available.

Apache 2.0 mod_isapi may support additional features introduced in later versions of the ISAPI specification, as well as limited emulation of async I/O and the TransmitFile semantics. Apache 2.0 may also support caching of ISAPI .dlls for performance. No further enhancements to the Apache 1.3 mod_isapi features are anticipated.

ISAPIReadAheadBuffer directive

Syntax: ISAPIReadAheadBuffer *size*

Default: 49152

Context: server config

Override: None

Status: Base

Module: mod_isapi

Compatibility: Apache 1.3.13 and later, Win32 only

Defines the maximum size of the Read Ahead Buffer sent to ISAPI extensions when they are initially invoked. All remaining data must be retrieved using the ReadClient callback; some ISAPI extensions may not support the ReadClient function. Refer questions to the ISAPI extension's author.

ISAPILogNotSupported directive

Syntax: ISAPILogNotSupported on|off

Default: on

Context: server config

Override: None

Status: Base

Module: mod_isapi

Compatibility: Apache 1.3.13 and later, Win32 only

Logs all requests for unsupported features from ISAPI extensions in the server error log. While this should be turned off once all desired ISAPI modules are functioning, it defaults to on to help administrators track down problems.

ISAPIAppendLogToErrors directive

Syntax: ISAPIAppendLogToErrors on|off

Default: off

Context: server config

Override: None

Status: Base

Module: mod_isapi

Compatibility: Apache 1.3.13 and later, Win32 only

Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extentions to the server error log.

ISAPIAppendLogToQuery directive

Syntax: ISAPIAppendLogToQuery on|off

Default: off

Context: server config

Override: None

Status: Base

Module: mod_isapi

Compatibility: Apache 1.3.13 and later, Win32 only

Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extentions to the query field (appended to the CustomLog %q component).

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_log_agent

This module provides for logging of the client user agents.

Status: Extension

Source File: mod_log_agent.c

Module Identifier: agent_log_module

Summary

This module is provided strictly for compatibility with NCSA httpd, and is deprecated. We recommend you use [mod_log_config](#) instead.

Directives

- [AgentLog](#)

See also: [CustomLog](#) and [LogFormat](#).

AgentLog

Syntax: AgentLog *file-pipe*

Default: AgentLog logs/agent_log

Context: server config, virtual host

Status: Extension

Module: mod_log_agent

The AgentLog directive sets the name of the file to which the server will log the UserAgent header of incoming requests. *File-pipe* is one of

A filename

A filename relative to the [ServerRoot](#).

`|' followed by a command

A program to receive the agent log information on its standard input. Note the a new program will not be started for a VirtualHost if it inherits the AgentLog from the main server.

Security: if a program is used, then it will be run under the user who started httpd. This will be root if the server was started by root; be sure that the program is secure.

Security: See the [security tips](#) document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

This directive is provided for compatibility with NCSA 1.4.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_log_config

This module provides for logging of the requests made to the server, using the Common Log Format or a user-specified format.

Status: Base

Source File: mod_log_config.c

Module Identifier: config_log_module

Compatibility: Was an extension module prior to Apache 1.2.

Summary

This module provides for flexible logging of client requests. Logs are written in a customizable format, and may be written directly to a file, or to an external program. Conditional logging is provided so that individual requests may be included or excluded from the logs based on characteristics of the request.

Three directives are provided by this module: `TransferLog` to create a log file, `LogFormat` to set a custom format, and `CustomLog` to define a log file and format in one step. The `TransferLog` and `CustomLog` directives can be used multiple times in each server to cause each request to be logged to multiple files.

See also: [Apache Log Files](#).

Directives

- [CookieLog](#)
- [CustomLog](#)
- [LogFormat](#)
- [TransferLog](#)

Custom Log Formats

The format argument to the `LogFormat` and `CustomLog` directives is a string. This string is logged to the log file for each request. It can contain literal characters copied into the log files and the c-type control characters `"\n"` and `"\t"` to represent new-lines and tabs. Literal quotes and back-slashes should be escaped with back-slashes.

The characteristics of the request itself are logged by placing `"%"` directives in the format string, which are replaced in the log file by the values as follows:

```
%...a:      Remote IP-address
%...A:      Local IP-address
%...B:      Bytes sent, excluding HTTP headers.
%...b:      Bytes sent, excluding HTTP headers. In CLF format
            i.e. a '-' rather than a 0 when no bytes are sent.
%...c:      Connection status when response is completed.
            'X' = connection aborted before the response completed.
            '+' = connection may be kept alive after the response is sent.
```

```

    '-' = connection will be closed after the response is sent.
%...{FOOBAR}e:  The contents of the environment variable FOOBAR
%...f:         Filename
%...h:         Remote host
%...H          The request protocol
%...{Foobar}i: The contents of Foobar: header line(s) in the request
                sent to the server.
%...l:         Remote logname (from identd, if supplied)
%...m          The request method
%...{Foobar}n: The contents of note "Foobar" from another module.
%...{Foobar}o: The contents of Foobar: header line(s) in the reply.
%...p:         The canonical Port of the server serving the request
%...P:         The process ID of the child that serviced the request.
%...q          The query string (prepended with a ? if a query string exists,
                otherwise an empty string)
%...r:         First line of request
%...s:         Status.  For requests that got internally redirected, this is
                the status of the *original* request --- %...>s for the last.
%...t:         Time, in common log format time format (standard english format)
%...{format}t: The time, in the form given by format, which should
                be in strftime(3) format. (potentially localized)
%...T:         The time taken to serve the request, in seconds.
%...u:         Remote user (from auth; may be bogus if return status (%s) is 401)
%...U:         The URL path requested, not including any query string.
%...v:         The canonical ServerName of the server serving the request.
%...V:         The server name according to the UseCanonicalName setting.

```

The "..." can be nothing at all (e.g., "%h %u %r %s %b"), or it can indicate conditions for inclusion of the item (which will cause it to be replaced with "-" if the condition is not met). The forms of condition are a list of HTTP status codes, which may or may not be preceded by "!". Thus, "%400,501{User-agent}i" logs User-agent: on 400 errors and 501 errors (Bad Request, Not Implemented) only; "%!200,304,302{Referer}i" logs Referer: on all requests which did **not** return some sort of normal status.

Note that there is no escaping performed on the strings from %...r, %...i and %...o. This is mainly to comply with the requirements of the Common Log Format. This implies that clients can insert control characters into the log, so care should be taken when dealing with raw log files.

Some commonly used log format strings are:

Common Log Format (CLF)

```
"%h %l %u %t \"%r\" %>s %b"
```

Common Log Format with Virtual Host

```
"%v %h %l %u %t \"%r\" %>s %b"
```

NCSA extended/combined log format

```
"%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
```

Referer log format

```
"%{Referer}i -> %U"
```

Agent (Browser) log format

```
"%{User-agent}i"
```

Note that the canonical [ServerName](#) and [Port](#) of the server serving the request are used for %v and %p respectively. This happens regardless of the [UseCanonicalName](#) setting because otherwise log analysis programs would have to duplicate the entire vhost matching algorithm in order to decide what host really served the request.

Security Considerations

See the [security tips](#) document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

Compatibility notes

- This module is based on mod_log_config distributed with previous Apache releases, now updated to handle multiple logs. There is now no need to re-configure Apache to use configuration log formats.
 - The module also implements the CookieLog directive, used to log user-tracking information created by [mod_usertrack](#). The use of CookieLog is deprecated, and a CustomLog should be defined to log user-tracking information instead.
 - As of Apache 1.3.5, this module allows conditional logging based upon the setting of [environment variables](#). That is, you can control whether a request should be logged or not based upon whether an arbitrary environment variable is defined or not. This is configurable on a *per*-logfile basis.
 - Beginning with Apache 1.3.5, the mod_log_config module has also subsumed the RefererIgnore functionality from [mod_log_referer](#). The effect of RefererIgnore can be achieved by combinations of [SetEnvIf](#) directives and conditional CustomLog definitions.
-

CookieLog directive

Syntax: CookieLog *filename*

Context: server config, virtual host

Module: mod_cookies

Compatibility: Only available in Apache 1.2 and above

The CookieLog directive sets the filename for logging of cookies. The filename is relative to the [ServerRoot](#). This directive is included only for compatibility with [mod_cookies](#), and is deprecated.

CustomLog directive

Syntax: CustomLog *file|pipe format|nickname* [env=[!]*environment-variable*]

Context: server config, virtual host

Status: Base

Compatibility: Nickname only available in Apache 1.3 or later. Conditional logging available in 1.3.5 or later.

Module: mod_log_config

The CustomLog directive is used to log requests to the server. A log format is specified, and the logging can optionally be made conditional on request characteristics using environment variables.

The first argument, which specifies the location to which the logs will be written, can take on one of the following two types of values:

file

A filename, relative to the [ServerRoot](#).

pipe

The pipe character "|", followed by the path to a program to receive the log information on its standard input. **Security:** if a program is used, then it will be run under the user who started httpd. This will be root if the server was started by root; be sure that the program is secure.

The second argument specifies what will be written to the log file. It can specify either a *nickname* defined by a previous [LogFormat](#)

directive, or it can be an explicit *format* string as described in the [log formats](#) section.

For example, the following two sets of directives have exactly the same effect:

```
# CustomLog with format nickname
LogFormat "%h %l %u %t \"%r\" %>s %b" common
CustomLog logs/access_log common

# CustomLog with explicit format string
CustomLog logs/access_log "%h %l %u %t \"%r\" %>s %b"
```

The third argument is optional and allows the decision on whether or not to log a particular request to be based on the presence or absence of a particular variable in the server environment. If the specified [environment variable](#) is set for the request (or is not set, in the case of a 'env=!name' clause), then the request will be logged.

Environment variables can be set on a *per-request* basis using the [mod_setenvif](#) and/or [mod_rewrite](#) modules. For example, if you don't want to record requests for all GIF images on your server in a separate logfile but not your main log, you can use:

```
SetEnvIf Request_URI \.gif$ gif-image
CustomLog gif-requests.log common env=gif-image
CustomLog nongif-requests.log common env=!gif-image
```

LogFormat directive

Syntax: LogFormat *format*/*nickname* [*nickname*]

Default: LogFormat "%h %l %u %t \"%r\" %>s %b"

Context: server config, virtual host

Status: Base

Compatibility: Nickname only available in Apache 1.3 or later

Module: mod_log_config

This directive specifies the format of the access log file.

The LogFormat directive can take one of two forms. In the first form, where only one argument is specified, this directive sets the log format which will be used by logs specified in subsequent [TransferLog](#) directives. The single argument can specify an explicit *format* as discussed in [custom log formats](#) section above. Alternatively, it can use a *nickname* to refer to a log format defined in a previous LogFormat directive as described below.

The second form of the LogFormat directive associates an explicit *format* with a *nickname*. This *nickname* can then be used in subsequent LogFormat or [CustomLog](#) directives rather than repeating the entire format string. A LogFormat directive which defines a nickname **does nothing else** -- that is, it *only* defines the nickname, it doesn't actually apply the format and make it the default. Therefore, it will not affect subsequent [TransferLog](#) directives.

TransferLog directive

Syntax: TransferLog *file*|*pipe*

Default: none

Context: server config, virtual host

Status: Base

Module: mod_log_config

This directive has exactly the same arguments and effect as the [CustomLog](#) directive, with the exception that it does not allow the log format to be specified explicitly or for conditional logging of requests. Instead, the log format is determined by the most recently

specified [LogFormat](#) directive (that does not define a nickname). Common Log Format is used if no other format has been specified.

Example:

```
LogFormat "%h %l %u %t \"%r\" %>s %b \"%{Referer}i\" \"%{User-agent}i\""
TransferLog logs/access_log
```

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_log_referer

This module provides for logging of the documents which reference documents on the server.

Status: Extension

Source File: mod_log_referer.c

Module Identifier: referer_log_module

Summary

This module is provided strictly for compatibility with NCSA httpd, and is deprecated. We recommend you use [mod_log_config](#) instead.

Directives

- [RefererIgnore](#)
- [RefererLog](#)

See also: [CustomLog](#) and [LogFormat](#).

Log file format

The log file contains a separate line for each refer. Each line has the format

uri -> document

where *uri* is the (%-escaped) URI for the document that references the one requested by the client, and *document* is the (%-decoded) local URL to the document being referred to.

RefererIgnore directive

Syntax: RefererIgnore *string* [*string*] ...

Context: server config, virtual host

Status: Extension

Module: mod_log_referer

The RefererIgnore directive adds to the list of strings to ignore in Referer headers. If any of the strings in the list is contained in the Referer header, then no referrer information will be logged for the request. Example:

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

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

RefererLog directive

Syntax: RefererLog *file-pipe*

Default: RefererLog logs/referer_log

Context: server config, virtual host

Status: Extension

Module: mod_log_referer

The RefererLog directive sets the name of the file to which the server will log the Referer header of incoming requests. *File-pipe* is one of

A filename

A filename relative to the [ServerRoot](#).

`|' followed by a command

A program to receive the referrer log information on its standard input. Note the a new program will not be started for a VirtualHost if it inherits the RefererLog from the main server.

Security: if a program is used, then it will be run under the user who started httpd. This will be root if the server was started by root; be sure that the program is secure.

Security: See the [security tips](#) document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

This directive is provided for compatibility with NCSA 1.4.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_mime

This module provides for determining the types of files from the filename and for association of handlers with files.

Status: Base

Source File: mod_mime.c

Module Identifier: mime_module

Summary

This module is used to determine various bits of "meta information" about documents. This information relates to the content of the document and is returned to the browser or used in content-negotiation within the server. In addition, a "handler" can be set for a document, which determines how the document will be processed within the server.

The directives [AddCharset](#), [AddEncoding](#), [AddHandler](#), [AddLanguage](#) and [AddType](#) are all used to map file extensions onto the meta-information for that file. Respectively they set the character set, content-encoding, handler, content-language, and MIME-type (content-type) of documents. The directive [TypesConfig](#) is used to specify a file which also maps extensions onto MIME types. The directives [ForceType](#) and [SetHandler](#) are used to associated all the files in a given location (*e.g.*, a particular directory) onto a particular MIME type or handler.

Note that changing the type or encoding of a file does not change the value of the `Last-Modified` header. Thus, previously cached copies may still be used by a client or proxy, with the previous headers.

Directives

- [AddCharset](#)
- [AddEncoding](#)
- [AddHandler](#)
- [AddLanguage](#)
- [AddType](#)
- [DefaultLanguage](#)
- [ForceType](#)
- [RemoveEncoding](#)
- [RemoveHandler](#)
- [RemoveType](#)
- [SetHandler](#)
- [TypesConfig](#)

See also: [MimeMagicFile](#).

Files with Multiple Extensions

Files can have more than one extension, and the order of the extensions is *normally* irrelevant. For example, if the file `welcome.html.fr` maps onto content type `text/html` and language French then the file `welcome.fr.html` will map onto exactly the same information. The only exception to this is if an extension is given which Apache does not know how to handle. In this case it will "forget" about any information it obtained from extensions to the left of the unknown extension. So, for example, if the extensions `fr` and `html` are mapped to the appropriate language and type but extension `xxx` is not assigned to anything, then the file `welcome.fr.xxx.html` will be associated with content-type `text/html` but *no* language.

If more than one extension is given which maps onto the same type of meta-information, then the one to the right will be used. For example, if `.gif` maps to the MIME-type `image/gif` and `.html` maps to the MIME-type `text/html`, then the file `welcome.gif.html` will be associated with the MIME-type `text/html`.

Care should be taken when a file with multiple extensions gets associated with both a MIME-type and a handler. This will usually result in the request being by the module associated with the handler. For example, if the `.imap` extension is mapped to the handler "imap-file" (from `mod_imap`) and the `.html` extension is mapped to the MIME-type `text/html`, then the file `world.imap.html` will be associated with both the "imap-file" handler and `text/html` MIME-type. When it is processed, the "imap-file" handler will be used, and so it will be treated as a `mod_imap` imagemap file.

AddCharset directive

Syntax: `AddCharset charset extension [extension] ...`

Context: server config, virtual host, directory, `.htaccess`

Override: FileInfo

Status: Base

Module: `mod_mime`

Compatibility: AddCharset is only available in Apache 1.3.10 and later

The AddCharset directive maps the given filename extensions to the specified content charset. *charset* is the MIME charset parameter of filenames containing *extension*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*.

Example:

```
AddLanguage ja .ja
AddCharset EUC-JP .euc
AddCharset ISO-2022-JP .jis
AddCharset SHIFT_JIS .sjis
```

Then the document `xxxx.ja.jis` will be treated as being a Japanese document whose charset is ISO-2022-JP (as will the document `xxxx.jis.ja`). The AddCharset directive is useful for both to inform the client about the character encoding of the document so that the document can be interpreted and displayed appropriately, and for [content negotiation](#), where the server returns one from several documents based on the client's charset preference.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

See also: [mod_negotiation](#)

AddEncoding directive

Syntax: `AddEncoding MIME-enc extension [extension] ...`

Context: server config, virtual host, directory, `.htaccess`

Override: FileInfo

Status: Base

Module: mod_mime

The `AddEncoding` directive maps the given filename extensions to the specified encoding type. *MIME-enc* is the MIME encoding to use for documents containing the *extension*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*. Example:

```
AddEncoding x-gzip .gz
AddEncoding x-compress .Z
```

This will cause filenames containing the `.gz` extension to be marked as encoded using the `x-gzip` encoding, and filenames containing the `.Z` extension to be marked as encoded with `x-compress`.

Old clients expect `x-gzip` and `x-compress`, however the standard dictates that they're equivalent to `gzip` and `compress` respectively. Apache does content encoding comparisons by ignoring any leading `x-`. When responding with an encoding Apache will use whatever form (*i.e.*, `x-foo` or `foo`) the client requested. If the client didn't specifically request a particular form Apache will use the form given by the `AddEncoding` directive. To make this long story short, you should always use `x-gzip` and `x-compress` for these two specific encodings. More recent encodings, such as `deflate` should be specified without the `x-`.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

See also: [Files with multiple extensions](#)

AddHandler directive

Syntax: `AddHandler handler-name extension [extension] ...`

Context: server config, virtual host, directory, `.htaccess`

Override: FileInfo

Status: Base

Module: mod_mime

Compatibility: AddHandler is only available in Apache 1.1 and later

AddHandler maps the filename extensions *extension* to the [handler](#) *handler-name*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*. For example, to activate CGI scripts with the file extension `".cgi"`, you might use:

```
AddHandler cgi-script .cgi
```

Once that has been put into your `srm.conf` or `httpd.conf` file, any file containing the `".cgi"` extension will be treated as a CGI program.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

See also: [Files with multiple extensions](#)

AddLanguage directive

Syntax: `AddLanguage MIME-lang extension [extension] ...`

Context: server config, virtual host, directory, `.htaccess`

Override: FileInfo

Status: Base

Module: mod_mime

The AddLanguage directive maps the given filename extension to the specified content language. *MIME-lang* is the MIME language of filenames containing *extension*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*.

Example:

```
AddEncoding x-compress .Z
AddLanguage en .en
AddLanguage fr .fr
```

Then the document `xxxx.en.Z` will be treated as being a compressed English document (as will the document `xxxx.Z.en`). Although the content language is reported to the client, the browser is unlikely to use this information. The `AddLanguage` directive is more useful for [content negotiation](#), where the server returns one from several documents based on the client's language preference.

If multiple language assignments are made for the same extension, the last one encountered is the one that is used. That is, for the case of:

```
AddLanguage en .en
AddLanguage en-uk .en
AddLanguage en-us .en
```

documents with the extension ".en" would be treated as being "en-us".

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

See also: [Files with multiple extensions](#)

See also: [mod_negotiation](#)

AddType directive

Syntax: `AddType MIME-type extension [extension] ...`

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_mime

The `AddType` directive maps the given filename extensions onto the specified content type. *MIME-type* is the MIME type to use for filenames containing *extension*. This mapping is added to any already in force, overriding any mappings that already exist for the same *extension*. This directive can be used to add mappings not listed in the MIME types file (see the [TypesConfig](#) directive).

Example:

```
AddType image/gif .gif
```

It is recommended that new MIME types be added using the `AddType` directive rather than changing the [TypesConfig](#) file.

Note that, unlike the NCSA httpd, this directive cannot be used to set the type of particular files.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

See also: [Files with multiple extensions](#)

DefaultLanguage directive

Syntax: `DefaultLanguage MIME-lang`

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_mime

Compatibility: `DefaultLanguage` is only available in Apache 1.3.4 and later.

The `DefaultLanguage` directive tells Apache that all files in the directive's scope (*e.g.*, all files covered by the current `<Directory>` container) that don't have an explicit language extension (such as `.fr` or `.de` as configured by `AddLanguage`) should be considered to be in the specified *MIME-lang* language. This allows entire directories to be marked as containing Dutch content, for instance, without having to rename each file. Note that unlike using extensions to specify languages, `DefaultLanguage` can only specify a single language.

If no `DefaultLanguage` directive is in force, and a file does not have any language extensions as configured by `AddLanguage`, then that file will be considered to have no language attribute.

See also: [mod_negotiation](#)

See also: [Files with multiple extensions](#)

ForceType directive

Syntax: `ForceType media-type`

Context: directory, `.htaccess`

Status: Base

Module: `mod_mime`

Compatibility: `ForceType` is only available in Apache 1.1 and later.

When placed into an `.htaccess` file or a `<Directory>` or `<Location>` section, this directive forces all matching files to be served as the content type given by *media type*. For example, if you had a directory full of GIF files, but did not want to label them all with `.gif`, you might want to use:

```
ForceType image/gif
```

Note that this will override any filename extensions that might determine the media type.

RemoveEncoding directive

Syntax: `RemoveEncoding extension [extension] ...`

Context: directory, `.htaccess`

Status: Base

Module: `mod_mime`

Compatibility: `RemoveEncoding` is only available in Apache 1.3.13 and later.

The `RemoveEncoding` directive removes any encoding associations for files with the given extensions. This allows `.htaccess` files in subdirectories to undo any associations inherited from parent directories or the server config files. An example of its use might be:

```
/foo/.htaccess:
    AddEncoding x-gzip .gz
    AddType text/plain .asc
    <Files *.gz.asc>
        RemoveEncoding .gz
    </Files>
```

This will cause `foo.gz` to mark as being encoded with the `gzip` method, but `foo.gz.asc` as an unencoded plaintext file.

Note: `RemoveEncoding` directives are processed *after* any `AddEncoding` directives, so it is possible they may undo the effects of the latter if both occur within the same directory configuration.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

RemoveHandler directive

Syntax: RemoveHandler *extension* [*extension*] ...

Context: directory, .htaccess

Status: Base

Module: mod_mime

Compatibility: RemoveHandler is only available in Apache 1.3.4 and later.

The RemoveHandler directive removes any handler associations for files with the given extensions. This allows .htaccess files in subdirectories to undo any associations inherited from parent directories or the server config files. An example of its use might be:

```
/foo/.htaccess:
    AddHandler server-parsed .html

/foo/bar/.htaccess:
    RemoveHandler .html
```

This has the effect of returning .html files in the /foo/bar directory to being treated as normal files, rather than as candidates for parsing (see the [mod_include](#) module).

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

RemoveType directive

Syntax: RemoveType *extension* [*extension*] ...

Context: directory, .htaccess

Status: Base

Module: mod_mime

Compatibility: RemoveType is only available in Apache 1.3.13 and later.

The RemoveType directive removes any MIME type associations for files with the given extensions. This allows .htaccess files in subdirectories to undo any associations inherited from parent directories or the server config files. An example of its use might be:

```
/foo/.htaccess:
    RemoveType .cgi
```

This will remove any special handling of .cgi files in the /foo/ directory and any beneath it, causing the files to be treated as being of the [default type](#).

Note: RemoveType directives are processed *after* any AddType directives, so it is possible they may undo the effects of the latter if both occur within the same directory configuration.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

SetHandler directive

Syntax: SetHandler *handler-name*

Context: directory, .htaccess

Status: Base

Module: mod_mime

Compatibility: SetHandler is only available in Apache 1.1 and later.

When placed into an `.htaccess` file or a `<Directory>` or `<Location>` section, this directive forces all matching files to be parsed through the [handler](#) given by *handler-name*. For example, if you had a directory you wanted to be parsed entirely as imagemap rule files, regardless of extension, you might put the following into an `.htaccess` file in that directory:

```
SetHandler imap-file
```

Another example: if you wanted to have the server display a status report whenever a URL of `http://servername/status` was called, you might put the following into `access.conf`:

```
<Location /status>  
SetHandler server-status  
</Location>
```

TypesConfig directive

Syntax: `TypesConfig file-path`

Default: `TypesConfig conf/mime.types`

Context: server config

Status: Base

Module: mod_mime

The `TypesConfig` directive sets the location of the MIME types configuration file. *Filename* is relative to the [ServerRoot](#). This file sets the default list of mappings from filename extensions to content types; changing this file is not recommended. Use the [AddType](#) directive instead. The file contains lines in the format of the arguments to an `AddType` command:

MIME-type extension extension ...

The extensions are lower-cased. Blank lines, and lines beginning with a hash character (`#`) are ignored.

Apache HTTP Server Version 1.3





Module mod_mime_magic

This module provides for determining the MIME type of a file by looking at a few bytes of its contents.

Status: Extension

Source File: mod_mime_magic.c

Module Identifier: mime_magic_module

Summary

This module determines the MIME type of files in the same way the Unix `file(1)` command works: it looks at the first few bytes of the file. It is intended as a "second line of defense" for cases that [mod_mime](#) can't resolve. To assure that `mod_mime` gets first try at determining a file's MIME type, be sure to list `mod_mime_magic` **before** `mod_mime` in the configuration.

This module is derived from a free version of the `file(1)` command for Unix, which uses "magic numbers" and other hints from a file's contents to figure out what the contents are. This module is active only if the magic file is specified by the [MimeMagicFile](#) directive.

Directives

- [MimeMagicFile](#)

Format of the Magic File

The contents of the file are plain ASCII text in 4-5 columns. Blank lines are allowed but ignored. Commented lines use a hash mark "#". The remaining lines are parsed for the following columns:

Column	Description																						
1	byte number to begin checking from ">" indicates a dependency upon the previous non-">" line																						
2	type of data to match <table border="1"> <tbody> <tr> <td>byte</td> <td>single character</td> </tr> <tr> <td>short</td> <td>machine-order 16-bit integer</td> </tr> <tr> <td>long</td> <td>machine-order 32-bit integer</td> </tr> <tr> <td>string</td> <td>arbitrary-length string</td> </tr> <tr> <td>date</td> <td>long integer date (seconds since Unix epoch/1970)</td> </tr> <tr> <td>beshort</td> <td>big-endian 16-bit integer</td> </tr> <tr> <td>belong</td> <td>big-endian 32-bit integer</td> </tr> <tr> <td>bedate</td> <td>big-endian 32-bit integer date</td> </tr> <tr> <td>leshort</td> <td>little-endian 16-bit integer</td> </tr> <tr> <td>lelong</td> <td>little-endian 32-bit integer</td> </tr> <tr> <td>ledate</td> <td>little-endian 32-bit integer date</td> </tr> </tbody> </table>	byte	single character	short	machine-order 16-bit integer	long	machine-order 32-bit integer	string	arbitrary-length string	date	long integer date (seconds since Unix epoch/1970)	beshort	big-endian 16-bit integer	belong	big-endian 32-bit integer	bedate	big-endian 32-bit integer date	leshort	little-endian 16-bit integer	lelong	little-endian 32-bit integer	ledate	little-endian 32-bit integer date
byte	single character																						
short	machine-order 16-bit integer																						
long	machine-order 32-bit integer																						
string	arbitrary-length string																						
date	long integer date (seconds since Unix epoch/1970)																						
beshort	big-endian 16-bit integer																						
belong	big-endian 32-bit integer																						
bedate	big-endian 32-bit integer date																						
leshort	little-endian 16-bit integer																						
lelong	little-endian 32-bit integer																						
ledate	little-endian 32-bit integer date																						
3	contents of data to match																						

4	MIME type if matched
5	MIME encoding if matched (optional)

For example, the following magic file lines would recognize some audio formats.

```
# Sun/NeXT audio data
0      string      .snd
>12    belong      1          audio/basic
>12    belong      2          audio/basic
>12    belong      3          audio/basic
>12    belong      4          audio/basic
>12    belong      5          audio/basic
>12    belong      6          audio/basic
>12    belong      7          audio/basic
>12    belong      23         audio/x-adpcm
```

Or these would recognize the difference between "*.doc" files containing Microsoft Word or FrameMaker documents. (These are incompatible file formats which use the same file suffix.)

```
# Frame
0      string      \<MakerFile      application/x-frame
0      string      \<MIFFFile       application/x-frame
0      string      \<MakerDictionary application/x-frame
0      string      \<MakerScreenFon application/x-frame
0      string      \<MML           application/x-frame
0      string      \<Book          application/x-frame
0      string      \<Maker         application/x-frame
```

```
# MS-Word
0      string      \376\067\0\043      application/msword
0      string      \320\317\021\340\241\261 application/msword
0      string      \333\245-\0\0\0      application/msword
```

An optional MIME encoding can be included as a fifth column. For example, this can recognize gzipped files and set the encoding for them.

```
# gzip (GNU zip, not to be confused with [Info-ZIP/PKWARE] zip archiver)
0      string      \037\213          application/octet-stream      x-gzip
```

Performance Issues

This module is not for every system. If your system is barely keeping up with its load or if you're performing a web server benchmark, you may not want to enable this because the processing is not free.

However, an effort was made to improve the performance of the original file(1) code to make it fit in a busy web server. It was designed for a server where there are thousands of users who publish their own documents. This is probably very common on intranets. Many times, it's helpful if the server can make more intelligent decisions about a file's contents than the file name allows ...even if just to reduce the "why doesn't my page work" calls when users improperly name their own files. You have to decide if the extra work suits your environment.

When compiling an Apache server, this module should be at or near the top of the list of modules in the Configuration file. The modules are listed in increasing priority so that will mean this one is used only as a last resort, just like it was designed to.

Notes

The following notes apply to the mod_mime_magic module and are included here for compliance with contributors' copyright restrictions that require their acknowledgment.

```
/*
 * mod_mime_magic: MIME type lookup via file magic numbers
 * Copyright (c) 1996-1997 Cisco Systems, Inc.
 *
 * This software was submitted by Cisco Systems to the Apache Group in July
 * 1997. Future revisions and derivatives of this source code must
 * acknowledge Cisco Systems as the original contributor of this module.
 * All other licensing and usage conditions are those of the Apache Group.
 *
 * Some of this code is derived from the free version of the file command
 * originally posted to comp.sources.unix. Copyright info for that program
 * is included below as required.
 * -----
 * - Copyright (c) Ian F. Darwin, 1987. Written by Ian F. Darwin.
 *
 * This software is not subject to any license of the American Telephone and
 * Telegraph Company or of the Regents of the University of California.
 *
 * Permission is granted to anyone to use this software for any purpose on any
 * computer system, and to alter it and redistribute it freely, subject to
 * the following restrictions:
 *
 * 1. The author is not responsible for the consequences of use of this
 * software, no matter how awful, even if they arise from flaws in it.
 *
 * 2. The origin of this software must not be misrepresented, either by
 * explicit claim or by omission. Since few users ever read sources, credits
 * must appear in the documentation.
 *
 * 3. Altered versions must be plainly marked as such, and must not be
 * misrepresented as being the original software. Since few users ever read
 * sources, credits must appear in the documentation.
 *
 * 4. This notice may not be removed or altered.
 * -----
 *
 * For compliance with Mr Darwin's terms: this has been very significantly
 * modified from the free "file" command.
 * - all-in-one file for compilation convenience when moving from one
 * version of Apache to the next.
 * - Memory allocation is done through the Apache API's pool structure.
 * - All functions have had necessary Apache API request or server
 * structures passed to them where necessary to call other Apache API
 * routines. (i.e., usually for logging, files, or memory allocation in
 * itself or a called function.)
 * - struct magic has been converted from an array to a single-ended linked
 * list because it only grows one record at a time, it's only accessed
 * sequentially, and the Apache API has no equivalent of realloc().
 * - Functions have been changed to get their parameters from the server
 * configuration instead of globals. (It should be reentrant now but has
 * not been tested in a threaded environment.)
 * - Places where it used to print results to stdout now saves them in a
 * list where they're used to set the MIME type in the Apache request
 * record.
 * - Command-line flags have been removed since they will never be used here.
 */
```

MimeMagicFile

Syntax: MimeMagicFile *file-path*

Default: none

Context: server config, virtual host

Status: Extension

Module: mod_mime_magic

The MimeMagicFile directive can be used to enable this module, the default file is distributed at `conf/magic`. Non-rooted paths are relative to the `ServerRoot`. Virtual hosts will use the same file as the main server unless a more specific setting is used, in which case the more specific setting overrides the main server's file.



Apache HTTP Server Version 1.3

Module mod_mmap_static

This module provides mmap()ing of a statically configured list of frequently requested but not changed files.

Status: Experimental

Source File: mod_mmap_static.c

Module Identifier: mmap_static_module

Compatibility: Available in Apache 1.3 and later.

Summary

This is an **experimental** module and should be used with care. You can easily create a broken site using this module, read this document carefully. `mod_mmap_static` maps a list of statically configured files (via `MMapFile` directives in the main server configuration) into memory through the system call `mmap()`. This system call is available on most modern Unix derivatives, but not on all. There are sometimes system-specific limits on the size and number of files that can be `mmap()`d, experimentation is probably the easiest way to find out.

This `mmap()`ing is done once at server start or restart, only. So whenever one of the mapped files changes on the filesystem you *have* to restart the server by at least sending it a HUP or USR1 signal (see the [Stopping and Restarting](#) documentation). To reiterate that point: if the files are modified *in place* without restarting the server you may end up serving requests that are completely bogus. You should update files by unlinking the old copy and putting a new copy in place. Most tools such as `rdist` and `mv` do this. The reason why this module doesn't take care of changes to the files is that this check would need an extra `stat()` every time which is a waste and against the intent of I/O reduction.

Directives

- [MMapFile](#)

MMapFile directive

Syntax: `MMapFile filename [filename] ...`

Default: *None*

Context: server-config

Override: *Not applicable*

Status: Experimental

Module: mod_mmap_static

Compatibility: Only available in Apache 1.3 or later

The `MMapFile` directive maps one or more files (given as whitespace separated arguments) into memory at server startup time. They are automatically unmapped on a server shutdown. When the files have changed on the filesystem at least a HUP or USR1 signal should be sent to the server to re-mmap them.

Be careful with the *filename* arguments: They have to literally match the filesystem path Apache's URL-to-filename translation handlers create. We cannot compare inodes or other stuff to match paths through symbolic links *etc.* because that again would cost extra `stat()` system calls which is not acceptable. This module may or may not work with filenames rewritten by `mod_alias` or

mod_rewrite... it is an experiment after all.

Notice: You cannot use this for speeding up CGI programs or other files which are served by special content handlers. It can only be used for regular files which are usually served by the Apache core content handler.

Example:

```
MMapFile /usr/local/apache/htdocs/index.html
```

Note: don't bother asking for a for a MMapDir directive which recursively maps all the files in a directory. Use Unix the way it was meant to be used. For example, see the [Include](#) directive, and consider this command:

```
find /www/htdocs -type f -print \  
| sed -e 's/./mmapfile &/' > /www/conf/mmap.conf
```

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_negotiation

This module provides for [content negotiation](#).

Status: Base

Source File: mod_negotiation.c

Module Identifier: negotiation_module

Summary

Content negotiation, or more accurately content selection, is the selection of the document that best matches the clients capabilities, from one of several available documents. There are two implementations of this.

- A type map (a file with the handler `type-map`) which explicitly lists the files containing the variants.
- A MultiViews search (enabled by the MultiViews [Option](#), where the server does an implicit filename pattern match, and choose from amongst the results.

Directives

- [CacheNegotiatedDocs](#)
- [LanguagePriority](#)

See also: [DefaultLanguage](#), [AddEncoding](#), [AddLanguage](#), [AddType](#), and [Options](#).

Type maps

A type map has the same format as RFC822 mail headers. It contains document descriptions separated by blank lines, with lines beginning with a hash character ('#') treated as comments. A document description consists of several header records; records may be continued on multiple lines if the continuation lines start with spaces. The leading space will be deleted and the lines concatenated. A header record consists of a keyword name, which always ends in a colon, followed by a value. Whitespace is allowed between the header name and value, and between the tokens of value. The headers allowed are:

Content-Encoding:

The encoding of the file. Apache only recognizes encodings that are defined by an [AddEncoding](#) directive. This normally includes the encodings `x-compress` for compress'd files, and `x-gzip` for gzip'd files. The `x-` prefix is ignored for encoding comparisons.

Content-Language:

The language of the variant, as an Internet standard language tag (RFC 1766). An example is `en`, meaning English.

Content-Length:

The length of the file, in bytes. If this header is not present, then the actual length of the file is used.

Content-Type:

The MIME media type of the document, with optional parameters. Parameters are separated from the media type and from one another by a semi-colon, with a syntax of `name=value`. Common parameters include:

level

an integer specifying the version of the media type. For `text/html` this defaults to 2, otherwise 0.

qs

a floating-point number with a value in the range 0.0 to 1.0, indicating the relative 'quality' of this variant compared to the other available variants, independent of the client's capabilities. For example, a jpeg file is usually of higher source quality than an ascii file if it is attempting to represent a photograph. However, if the resource being represented is ascii art, then an ascii file would have a higher source quality than a jpeg file. All qs values are therefore specific to a given resource.

Example:

```
Content-Type: image/jpeg; qs=0.8
```

URI:

The path to the file containing this variant, relative to the map file.

MultiViews

A MultiViews search is enabled by the MultiViews [Option](#). If the server receives a request for `/some/dir/foo` and `/some/dir/foo` does *not* exist, then the server reads the directory looking for all files named `foo.*`, and effectively fakes up a type map which names all those files, assigning them the same media types and content-encodings it would have if the client had asked for one of them by name. It then chooses the best match to the client's requirements, and returns that document.

CacheNegotiatedDocs directive

Syntax: CacheNegotiatedDocs

Context: server config

Status: Base

Module: mod_negotiation

Compatibility: CacheNegotiatedDocs is only available in Apache 1.1 and later.

If set, this directive allows content-negotiated documents to be cached by proxy servers. This could mean that clients behind those proxys could retrieve versions of the documents that are not the best match for their abilities, but it will make caching more efficient.

This directive only applies to requests which come from HTTP/1.0 browsers. HTTP/1.1 provides much better control over the caching of negotiated documents, and this directive has no effect in responses to HTTP/1.1 requests.

LanguagePriority directive

Syntax: LanguagePriority *MIME-lang* [*MIME-lang*] ...

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_negotiation

The LanguagePriority sets the precedence of language variants for the case where the client does not express a preference, when handling a MultiViews request. The list of *MIME-lang* are in order of decreasing preference. Example:

```
LanguagePriority en fr de
```

For a request for `foo.html`, where `foo.html.fr` and `foo.html.de` both existed, but the browser did not express a language preference, then `foo.html.fr` would be returned.

Note that this directive only has an effect if a 'best' language cannot be determined by any other means. Correctly implemented HTTP/1.1 requests will mean this directive has no effect.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache module mod_proxy

This module provides for an **HTTP 1.0** caching proxy server.

Status: Extension

Source File: mod_proxy.c

Module Identifier: proxy_module

Compatibility: Available in Apache 1.1 and later.

Summary

This module implements a proxy/cache for Apache. It implements proxying capability for FTP, CONNECT (for SSL), HTTP/0.9, and HTTP/1.0. The module can be configured to connect to other proxy modules for these and other protocols.

This module was experimental in Apache 1.1.x. As of Apache 1.2, mod_proxy stability is *greatly* improved.

Directives

- [ProxyRequests](#)
- [ProxyRemote](#)
- [ProxyPass](#)
- [ProxyPassReverse](#)
- [ProxyBlock](#)
- [AllowCONNECT](#)
- [ProxyReceiveBufferSize](#)
- [NoProxy](#)
- [ProxyDomain](#)
- [ProxyVia](#)
- [CacheRoot](#)
- [CacheSize](#)
- [CacheMaxExpire](#)
- [CacheDefaultExpire](#)
- [CacheLastModifiedFactor](#)
- [CacheGcInterval](#)
- [CacheDirLevels](#)
- [CacheDirLength](#)
- [CacheForceCompletion](#)

- [NoCache](#)

Common configuration topics

- [Controlling access to your proxy](#)
- [Using Netscape hostname shortcuts](#)
- [Why doesn't file type xxx download via FTP?](#)
- [Why does Apache start more slowly when using the proxy module?](#)
- [Can I use the Apache proxy module with my SOCKS proxy?](#)
- [What other functions are useful for an intranet proxy server?](#)

Controlling access to your proxy

You can control who can access your proxy via the normal <Directory> control block using the following example:

```
<Directory proxy:*>
Order Deny,Allow
Deny from [machines you'd like *not* to allow by IP address or name]
Allow from [machines you'd like to allow by IP address or name]
</Directory>
```

A <Files> block will also work, and is the only method known to work for all possible URLs in Apache versions earlier than 1.2b10.

Using Netscape hostname shortcuts

There is an optional patch to the proxy module to allow Netscape-like hostname shortcuts to be used. It's available from the [contrib/patches/1.2](#) directory on the Apache Web site.

Why doesn't file type xxx download via FTP?

You probably don't have that particular file type defined as *application/octet-stream* in your proxy's mime.types configuration file. A useful line can be

```
application/octet-stream      bin dms lha lzh exe class tgz taz
```

How can I force an FTP ASCII download of File xxx?

In the rare situation where you must download a specific file using the FTP **ASCII** transfer method (while the default transfer is in **binary** mode), you can override mod_proxy's default by suffixing the request with ;type=a to force an ASCII transfer.

Why does Apache start more slowly when using the proxy module?

If you're using the ProxyBlock or NoCache directives, hostnames' IP addresses are looked up and cached during startup for later match test. This may take a few seconds (or more) depending on the speed with which the hostname lookups occur.

Can I use the Apache proxy module with my SOCKS proxy?

Yes. Just build Apache with the rule `SOCKS4=yes` in your *Configuration* file, and follow the instructions there. SOCKS5 capability can be added in a similar way (there's no `SOCKS5` rule yet), so use the `EXTRA_LDFLAGS` definition, or build Apache normally and run it with the *runsocks* wrapper provided with SOCKS5, if your OS supports dynamically linked libraries.

Some users have reported problems when using SOCKS version 4.2 on Solaris. The problem was solved by upgrading to SOCKS 4.3.

Remember that you'll also have to grant access to your Apache proxy machine by permitting connections on the appropriate ports in your SOCKS daemon's configuration.

What other functions are useful for an intranet proxy server?

An Apache proxy server situated in an intranet needs to forward external requests through the company's firewall. However, when it has to access resources within the intranet, it can bypass the firewall when accessing hosts. The [NoProxy](#) directive is useful for specifying which hosts belong to the intranet and should be accessed directly.

Users within an intranet tend to omit the local domain name from their WWW requests, thus requesting "http://somehost/" instead of "http://somehost.my.dom.ain/". Some commercial proxy servers let them get away with this and simply serve the request, implying a configured local domain. When the [ProxyDomain](#) directive is used and the server is [configured for proxy service](#), Apache can return a redirect response and send the client to the correct, fully qualified, server address. This is the preferred method since the user's bookmark files will then contain fully qualified hosts.

ProxyRequests directive

Syntax: ProxyRequests on|off

Default: ProxyRequests Off

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: ProxyRequests is only available in Apache 1.1 and later.

This allows or prevents Apache from functioning as a proxy server. Setting ProxyRequests to 'off' does not disable use of the [ProxyPass](#) directive.

ProxyRemote directive

Syntax: ProxyRemote *match remote-server*

Default: *None*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: ProxyRemote is only available in Apache 1.1 and later.

This defines remote proxies to this proxy. *match* is either the name of a URL-scheme that the remote server supports, or a partial URL for which the remote server should be used, or '*' to indicate the server should be contacted for all requests. *remote-server* is a partial URL for the remote server. Syntax:

```
remote-server = protocol://hostname[:port]
```

protocol is the protocol that should be used to communicate with the remote server; only "http" is supported by this module.

Example:

```
ProxyRemote http://goodguys.com/ http://mirrorguys.com:8000
ProxyRemote * http://cleversite.com
ProxyRemote ftp http://ftpproxy.mydomain.com:8080
```

In the last example, the proxy will forward FTP requests, encapsulated as yet another HTTP proxy request, to another proxy which can handle them.

ProxyPass directive

Syntax: ProxyPass *path url*

Default: None

Context: server config, virtual host

Override: Not applicable

Status: Base

Module: mod_proxy

Compatibility: ProxyPass is only available in Apache 1.1 and later.

This directive allows remote servers to be mapped into the space of the local server; the local server does not act as a proxy in the conventional sense, but appears to be a mirror of the remote server. *path* is the name of a local virtual path; *url* is a partial URL for the remote server.

Suppose the local server has address `http://wibble.org/`; then

```
ProxyPass /mirror/foo/ http://foo.com/
```

will cause a local request for the `<http://wibble.org/mirror/foo/bar>` to be internally converted into a proxy request to `<http://foo.com/bar>`.

ProxyPassReverse directive

Syntax: ProxyPassReverse *path url*

Default: None

Context: server config, virtual host

Override: Not applicable

Status: Base

Module: mod_proxy

Compatibility: ProxyPassReverse is only available in Apache 1.3b6 and later.

This directive lets Apache adjust the URL in the `Location` header on HTTP redirect responses. For instance this is essential when Apache is used as a reverse proxy to avoid by-passing the reverse proxy because of HTTP redirects on the backend servers which stay behind the reverse proxy.

path is the name of a local virtual path.

url is a partial URL for the remote server - the same way they are used for the `ProxyPass` directive.

Example:

Suppose the local server has address `http://wibble.org/`; then

```
ProxyPass          /mirror/foo/ http://foo.com/
ProxyPassReverse   /mirror/foo/ http://foo.com/
```

will not only cause a local request for the `<http://wibble.org/mirror/foo/bar>` to be internally converted into a proxy request to

<http://foo.com/bar> (the functionality ProxyPass provides here). It also takes care of redirects the server foo.com sends: when http://foo.com/bar is redirected by him to http://foo.com/quux Apache adjusts this to http://wibble.org/mirror/foo/quux before forwarding the HTTP redirect response to the client.

Note that this ProxyPassReverse directive can also be used in conjunction with the proxy pass-through feature ("RewriteRule ... [P]") from [mod_rewrite](#) because its doesn't depend on a corresponding ProxyPass directive.

AllowCONNECT directive

Syntax: AllowCONNECT *port* [*port*] ...

Default: AllowCONNECT 443 563

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: AllowCONNECT is only available in Apache 1.3.2 and later.

The AllowCONNECT directive specifies a list of port numbers to which the proxy CONNECT method may connect. Today's browsers use this method when a *https* connection is requested and proxy tunneling over *http* is in effect. By default, only the default https port (443) and the default snews port (563) are enabled. Use the AllowCONNECT directive to override this default and allow connections to the listed ports only.

ProxyBlock directive

Syntax: ProxyBlock *|*word/host/domain* [*word/host/domain*] ...

Default: *None*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: ProxyBlock is only available in Apache 1.2 and later.

The ProxyBlock directive specifies a list of words, hosts and/or domains, separated by spaces. HTTP, HTTPS, and FTP document requests to sites whose names contain matched words, hosts or domains are *blocked* by the proxy server. The proxy module will also attempt to determine IP addresses of list items which may be hostnames during startup, and cache them for match test as well. Example:

```
ProxyBlock joes-garage.com some-host.co.uk rocky.wotsamattau.edu
```

'rocky.wotsamattau.edu' would also be matched if referenced by IP address.

Note that 'wotsamattau' would also be sufficient to match 'wotsamattau.edu'.

Note also that

```
ProxyBlock *
```

blocks connections to all sites.

ProxyReceiveBufferSize directive

Syntax: ProxyReceiveBufferSize *bytes*

Default: *None*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: ProxyReceiveBufferSize is only available in Apache 1.3 and later.

The ProxyReceiveBufferSize directive specifies an explicit network buffer size for outgoing HTTP and FTP connections, for increased throughput. It has to be greater than 512 or set to 0 to indicate that the system's default buffer size should be used.

Example:

```
ProxyReceiveBufferSize 2048
```

NoProxy directive

Syntax: NoProxy *Domain|SubNet|IpAddr|Hostname* [*Domain|SubNet|IpAddr|Hostname*] ...

Default: *None*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: NoProxy is only available in Apache 1.3 and later.

This directive is only useful for Apache proxy servers within intranets. The NoProxy directive specifies a list of subnets, IP addresses, hosts and/or domains, separated by spaces. A request to a host which matches one or more of these is always served directly, without forwarding to the configured ProxyRemote proxy server(s).

Example:

```
ProxyRemote * http://firewall.mycompany.com:81
NoProxy .mycompany.com 192.168.112.0/21
```

The arguments to the NoProxy directive are one of the following type list:

Domain

A *Domain* is a partially qualified DNS domain name, preceded by a period. It represents a list of hosts which logically belong to the same DNS domain or zone (*i.e.*, the suffixes of the hostnames are all ending in *Domain*).

Examples: .com .apache.org.

To distinguish *Domains* from *Hostnames* (both syntactically and semantically; a DNS domain can have a DNS A record, too!), *Domains* are always written with a leading period.

Note: Domain name comparisons are done without regard to the case, and *Domains* are always assumed to be anchored in the root of the DNS tree, therefore two domains .MyDomain.com and .mydomain.com. (note the trailing period) are considered equal. Since a domain comparison does not involve a DNS lookup, it is much more efficient than subnet comparison.

SubNet

A *SubNet* is a partially qualified internet address in numeric (dotted quad) form, optionally followed by a slash and the netmask, specified as the number of significant bits in the *SubNet*. It is used to represent a subnet of hosts which can be reached over a common network interface. In the absence of the explicit net mask it is assumed that omitted (or zero valued) trailing digits specify the mask. (In this case, the netmask can only be multiples of 8 bits wide.)

Examples:

192.168 or 192.168.0.0

the subnet 192.168.0.0 with an implied netmask of 16 valid bits (sometimes used in the netmask form 255.255.0.0)
192.168.112.0/21

the subnet 192.168.112.0/21 with a netmask of 21 valid bits (also used in the form 255.255.248.0)

As a degenerate case, a *SubNet* with 32 valid bits is the equivalent to an *IPAddr*, while a *SubNet* with zero valid bits (e.g., 0.0.0.0/0) is the same as the constant *_Default_*, matching any IP address.

IPAddr

A *IPAddr* represents a fully qualified internet address in numeric (dotted quad) form. Usually, this address represents a host, but there need not necessarily be a DNS domain name connected with the address.

Example: 192.168.123.7

Note: An *IPAddr* does not need to be resolved by the DNS system, so it can result in more effective apache performance.

See Also: [DNS Issues](#)

Hostname

A *Hostname* is a fully qualified DNS domain name which can be resolved to one or more [IPAddrs](#) via the DNS domain name service. It represents a logical host (in contrast to [Domains](#), see above) and must be resolvable to at least one [IPAddr](#) (or often to a list of hosts with different [IPAddr](#)'s).

Examples: prep.ai.mit.edu www.apache.org.

Note: In many situations, it is more effective to specify an [IPAddr](#) in place of a *Hostname* since a DNS lookup can be avoided. Name resolution in Apache can take a remarkable deal of time when the connection to the name server uses a slow PPP link.

Note: *Hostname* comparisons are done without regard to the case, and *Hostnames* are always assumed to be anchored in the root of the DNS tree, therefore two hosts WWW.MyDomain.com and www.mydomain.com. (note the trailing period) are considered equal.

See Also: [DNS Issues](#)

ProxyDomain directive

Syntax: ProxyDomain *Domain*

Default: *None*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: ProxyDomain is only available in Apache 1.3 and later.

This directive is only useful for Apache proxy servers within intranets. The ProxyDomain directive specifies the default domain which the apache proxy server will belong to. If a request to a host without a domain name is encountered, a redirection response to the same host with the configured *Domain* appended will be generated.

Example:

```
ProxyRemote * http://firewall.mycompany.com:81
NoProxy .mycompany.com 192.168.112.0/21
ProxyDomain .mycompany.com
```

ProxyVia directive

Syntax: ProxyVia on|off|full|block

Default: *ProxyVia off*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: ProxyVia is only available in Apache 1.3.2 and later.

This directive controls the use of the Via: HTTP header by the proxy. Its intended use is to control the flow of proxy requests along a chain of proxy servers. See RFC2068 (HTTP/1.1) for an explanation of Via: header lines.

- If set to *off*, which is the default, no special processing is performed. If a request or reply contains a Via: header, it is passed through unchanged.
 - If set to *on*, each request and reply will get a Via: header line added for the current host.
 - If set to *full*, each generated Via: header line will additionally have the Apache server version shown as a Via: comment field.
 - If set to *block*, every proxy request will have all its Via: header lines removed. No new Via: header will be generated.
-

CacheForceCompletion directive

Syntax: CacheForceCompletion *percentage*

Default: 90

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheForceCompletion is only available in Apache 1.3.1 and later.

If an http transfer that is being cached is cancelled, the proxy module will complete the transfer to cache if more than the percentage specified has already been transferred.

This is a percentage, and must be a number between 1 and 100, or 0 to use the default. 100 will cause a document to be cached only if the transfer was allowed to complete. A number between 60 and 90 is recommended.

CacheRoot directive

Syntax: CacheRoot *directory*

Default: *None*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheRoot is only available in Apache 1.1 and later.

Sets the name of the directory to contain cache files; this must be writable by the httpd server. (see the [User](#) directive). Setting CacheRoot enables proxy cacheing; without defining a CacheRoot, proxy functionality will be available if ProxyRequests are set to On, but no cacheing will be available.

CacheSize directive

Syntax: CacheSize *kilobytes*

Default: CacheSize 5

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheSize is only available in Apache 1.1 and later.

Sets the desired space usage of the cache, in KB (1024-byte units). Although usage may grow above this setting, the garbage collection will delete files until the usage is at or below this setting.

Depending on the expected proxy traffic volume and CacheGcInterval, use a value which is at least 20 to 40 % lower than the available space.

CacheGcInterval directive

Syntax: CacheGcInterval *hours*

Default: *None*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheGcinterval is only available in Apache 1.1 and later.

Check the cache after the specified number of *hours*, and delete files if the space usage is greater than that set by CacheSize. Note that *hours* accepts a float value, you could for example use CacheGcInterval 1.5 to check the cache every 90 minutes. (If unset, no garbage collection will be performed, and the cache will grow indefinitely.) Note also that the larger the CacheGcInterval, the more extra space beyond the configured CacheSize will be needed for the cache between garbage collections.

CacheMaxExpire directive

Syntax: CacheMaxExpire *hours*

Default: CacheMaxExpire 24

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheMaxExpire is only available in Apache 1.1 and later.

Specifies the maximum number of *hours* for which cachable HTTP documents will be retained without checking the origin server. Thus, documents will be out of date at most this number of *hours*. This restriction is enforced even if an expiry date was supplied with the document.

CacheLastModifiedFactor directive

Syntax: CacheLastModifiedFactor *factor*

Default: CacheLastModifiedFactor 0.1

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheLastModifiedFactor is only available in Apache 1.1 and later.

If the origin HTTP server did not supply an expiry date for the document, then estimate one using the formula

$\text{expiry-period} = \text{time-since-last-modification} * \text{factor}$

For example, if the document was last modified 10 hours ago, and *factor* is 0.1, then the expiry period will be set to $10 * 0.1 = 1$ hour.

If the expiry-period would be longer than that set by CacheMaxExpire, then the latter takes precedence.

CacheDirLevels directive

Syntax: CacheDirLevels *levels*

Default: CacheDirLevels 3

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheDirLevels is only available in Apache 1.1 and later.

CacheDirLevels sets the number of *levels* of subdirectories in the cache. Cached data will be saved this many directory levels below CacheRoot.

CacheDirLength directive

Syntax: CacheDirLength *length*

Default: CacheDirLength 1

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheDirLength is only available in Apache 1.1 and later.

CacheDirLength sets the number of characters in proxy cache subdirectory names.

CacheDefaultExpire directive

Syntax: CacheDefaultExpire *hours*

Default: CacheDefaultExpire 1

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: CacheDefaultExpire is only available in Apache 1.1 and later.

If the document is fetched via a protocol that does not support expiry times, then use the specified number of *hours* as the expiry time. [CacheMaxExpire](#) does **not** override this setting.

NoCache directive

Syntax: NoCache *|*word/host/domain* [*word/host/domain*] ...

Default: *None*

Context: server config, virtual host

Override: *Not applicable*

Status: Base

Module: mod_proxy

Compatibility: NoCache is only available in Apache 1.1 and later.

The NoCache directive specifies a list of words, hosts and/or domains, separated by spaces. HTTP and non-passworded FTP documents from matched words, hosts or domains are *not* cached by the proxy server. The proxy module will also attempt to determine IP addresses of list items which may be hostnames during startup, and cache them for match test as well. Example:

```
NoCache joes-garage.com some-host.co.uk bullwinkle.wotsamattau.edu
```

'bullwinkle.wotsamattau.edu' would also be matched if referenced by IP address.

Note that 'wotsamattau' would also be sufficient to match 'wotsamattau.edu'.

Note also that

```
NoCache *
```

disables caching completely.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_rewrite URL Rewriting Engine

This module provides a rule-based rewriting engine to rewrite requested URLs on the fly.

Status: Extension

Source File: mod_rewrite.c

Module Identifier: rewrite_module

Compatibility: Available in Apache 1.2 and later.

Summary

``The great thing about mod_rewrite is it gives you all the configurability and flexibility of Sendmail. The downside to mod_rewrite is that it gives you all the configurability and flexibility of Sendmail.''

-- Brian Behlendorf
Apache Group

``Despite the tons of examples and docs, mod_rewrite is voodoo. Damned cool voodoo, but still voodoo.''

-- Brian Moore
bem@news.cmc.net

Welcome to mod_rewrite, the Swiss Army Knife of URL manipulation!

This module uses a rule-based rewriting engine (based on a regular-expression parser) to rewrite requested URLs on the fly. It supports an unlimited number of rules and an unlimited number of attached rule conditions for each rule to provide a really flexible and powerful URL manipulation mechanism. The URL manipulations can depend on various tests, for instance server variables, environment variables, HTTP headers, time stamps and even external database lookups in various formats can be used to achieve a really granular URL matching.

This module operates on the full URLs (including the path-info part) both in per-server context (`httpd.conf`) and per-directory context (`.htaccess`) and can even generate query-string parts on result. The rewritten result can lead to internal sub-processing, external request redirection or even to an internal proxy throughput.

But all this functionality and flexibility has its drawback: complexity. So don't expect to understand this entire module in just one day.

This module was invented and originally written in April 1996 and gifted exclusively to the The Apache Group in July 1997 by

[Ralf S. Engelschall](mailto:rse@engelschall.com)
rse@engelschall.com
www.engelschall.com

Table Of Contents

Internal Processing

- [API Phases](#)
- [Ruleset Processing](#)
- [Regex Back-Reference Availability](#)

Configuration Directives

- [RewriteEngine](#)
- [RewriteOptions](#)
- [RewriteLog](#)
- [RewriteLogLevel](#)
- [RewriteLock](#)
- [RewriteMap](#)
- [RewriteBase](#)
- [RewriteCond](#)
- [RewriteRule](#)

Miscellaneous

- [Environment Variables](#)
- [Practical Solutions](#)

Internal Processing

The internal processing of this module is very complex but needs to be explained once even to the average user to avoid common mistakes and to let you exploit its full functionality.

API Phases

First you have to understand that when Apache processes a HTTP request it does this in phases. A hook for each of these phases is provided by the Apache API. Mod_rewrite uses two of these hooks: the URL-to-filename translation hook which is used after the HTTP request has been read but before any authorization starts and the Fixup hook which is triggered after the authorization phases and after the per-directory config files (`.htaccess`) have been read, but before the content handler is activated.

So, after a request comes in and Apache has determined the corresponding server (or virtual server) the rewriting engine starts processing of all mod_rewrite directives from the per-server configuration in the URL-to-filename phase. A few steps later when the final data directories are found, the per-directory configuration directives of mod_rewrite are triggered in the Fixup phase. In both situations mod_rewrite rewrites URLs either to new URLs or to filenames, although there is no obvious distinction between them. This is a usage of the API which was not intended to be this way when the API was designed, but as of Apache 1.x this is the only way mod_rewrite can operate. To make this point more clear remember the following two points:

1. Although mod_rewrite rewrites URLs to URLs, URLs to filenames and even filenames to filenames, the API currently provides only a URL-to-filename hook. In Apache 2.0 the two missing hooks will be added to make the processing more clear. But this point has no drawbacks for the user, it is just a fact which should be remembered: Apache does more in the URL-to-filename hook than the API intends for it.
2. Unbelievably mod_rewrite provides URL manipulations in per-directory context, *i.e.*, within `.htaccess`

files, although these are reached a very long time after the URLs have been translated to filenames. It has to be this way because `.htaccess` files live in the filesystem, so processing has already reached this stage. In other words: According to the API phases at this time it is too late for any URL manipulations. To overcome this chicken and egg problem `mod_rewrite` uses a trick: When you manipulate a URL/filename in per-directory context `mod_rewrite` first rewrites the filename back to its corresponding URL (which is usually impossible, but see the `RewriteBase` directive below for the trick to achieve this) and then initiates a new internal sub-request with the new URL. This restarts processing of the API phases.

Again `mod_rewrite` tries hard to make this complicated step totally transparent to the user, but you should remember here: While URL manipulations in per-server context are really fast and efficient, per-directory rewrites are slow and inefficient due to this chicken and egg problem. But on the other hand this is the only way `mod_rewrite` can provide (locally restricted) URL manipulations to the average user.

Don't forget these two points!

Ruleset Processing

Now when `mod_rewrite` is triggered in these two API phases, it reads the configured rulesets from its configuration structure (which itself was either created on startup for per-server context or during the directory walk of the Apache kernel for per-directory context). Then the URL rewriting engine is started with the contained ruleset (one or more rules together with their conditions). The operation of the URL rewriting engine itself is exactly the same for both configuration contexts. Only the final result processing is different.

The order of rules in the ruleset is important because the rewriting engine processes them in a special (and not very obvious) order. The rule is this: The rewriting engine loops through the ruleset rule by rule (`RewriteRule` directives) and when a particular rule matches it optionally loops through existing corresponding conditions (`RewriteCond` directives). For historical reasons the conditions are given first, and so the control flow is a little bit long-winded. See Figure 1 for more details.

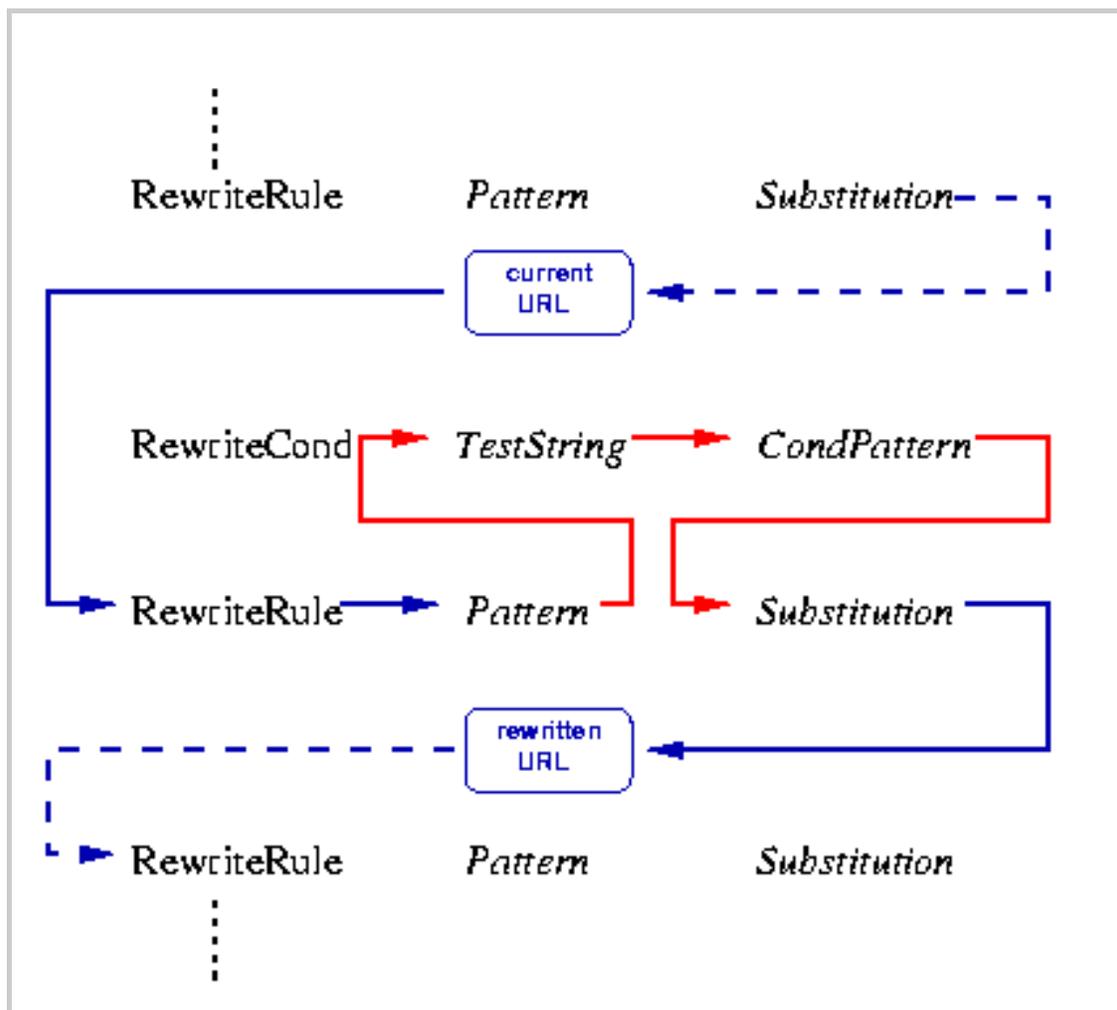


Figure 1: The control flow through the rewriting ruleset

As you can see, first the URL is matched against the *Pattern* of each rule. When it fails mod_rewrite immediately stops processing this rule and continues with the next rule. If the *Pattern* matches, mod_rewrite looks for corresponding rule conditions. If none are present, it just substitutes the URL with a new value which is constructed from the string *Substitution* and goes on with its rule-looping. But if conditions exist, it starts an inner loop for processing them in the order that they are listed. For conditions the logic is different: we don't match a pattern against the current URL. Instead we first create a string *TestString* by expanding variables, back-references, map lookups, *etc.* and then we try to match *CondPattern* against it. If the pattern doesn't match, the complete set of conditions and the corresponding rule fails. If the pattern matches, then the next condition is processed until no more conditions are available. If all conditions match, processing is continued with the substitution of the URL with *Substitution*.

Quoting Special Characters

As of Apache 1.3.20, special characters in *TestString* and *Substitution* strings can be escaped (that is, treated as normal characters without their usual special meaning) by prefixing them with a slosch ('\') character. In other words, you can include an actual dollar-sign character in a *Substitution* string by using '\\$'; this keeps mod_rewrite from trying to treat it as a backreference.

Regex Back-Reference Availability

One important thing here has to be remembered: Whenever you use parentheses in *Pattern* or in one of the *CondPattern*, back-references are internally created which can be used with the strings $\$N$ and $\%N$ (see below). These are available for creating the strings *Substitution* and *TestString*. Figure 2 shows to which locations the back-references are transferred for expansion.

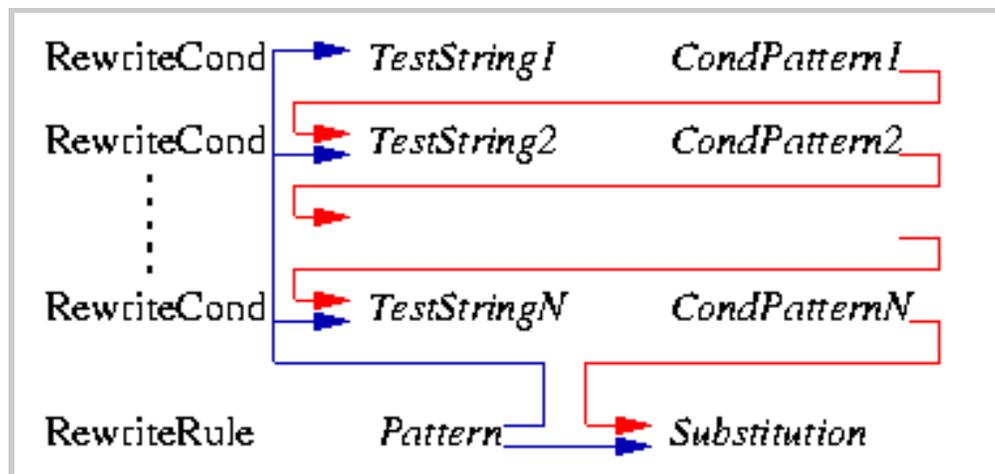


Figure 2: The back-reference flow through a rule

We know this was a crash course on mod_rewrite's internal processing. But you will benefit from this knowledge when reading the following documentation of the available directives.

Configuration Directives

RewriteEngine

Syntax: RewriteEngine on|off

Default: RewriteEngine off

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.2

The RewriteEngine directive enables or disables the runtime rewriting engine. If it is set to `off` this module does no runtime processing at all. It does not even update the SCRIPT_URx environment variables.

Use this directive to disable the module instead of commenting out all the RewriteRule directives!

Note that, by default, rewrite configurations are not inherited. This means that you need to have a RewriteEngine on directive for each virtual host in which you wish to use it.

RewriteOptions

Syntax: RewriteOptions *Option*

Default: None

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.2

The RewriteOptions directive sets some special options for the current per-server or per-directory configuration. The *Option* strings can be one of the following:

- **'inherit'**
This forces the current configuration to inherit the configuration of the parent. In per-virtual-server context this means that the maps, conditions and rules of the main server are inherited. In per-directory context this means that conditions and rules of the parent directory's .htaccess configuration are inherited.
-

RewriteLog

Syntax: RewriteLog *file-path*

Default: None

Context: server config, virtual host

Override: Not applicable

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.2

The RewriteLog directive sets the name of the file to which the server logs any rewriting actions it performs. If the name does not begin with a slash (/) then it is assumed to be relative to the *Server Root*. The directive should occur only once per server config.

Note: To disable the logging of rewriting actions it is not recommended to set *Filename* to `/dev/null`, because although the rewriting engine does not then output to a logfile it still creates the logfile output internally. **This will slow down the server with no advantage to the administrator!** To disable logging either remove or comment out the RewriteLog directive or use `RewriteLogLevel 0!`

Security: See the [Apache Security Tips](#) document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

Example:

```
RewriteLog "/usr/local/var/apache/logs/rewrite.log"
```

RewriteLogLevel

Syntax: RewriteLogLevel *Level*

Default: RewriteLogLevel 0

Context: server config, virtual host

Override: *Not applicable*

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.2

The RewriteLogLevel directive sets the verbosity level of the rewriting logfile. The default level 0 means no logging, while 9 or more means that practically all actions are logged.

To disable the logging of rewriting actions simply set *Level* to 0. This disables all rewrite action logs.

Notice: Using a high value for *Level* will slow down your Apache server dramatically!
Use the rewriting logfile at a *Level* greater than 2 only for debugging!

Example:

```
RewriteLogLevel 3
```

RewriteLock

Syntax: RewriteLock *file-path*

Default: *None*

Context: server config

Override: *Not applicable*

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.3

This directive sets the filename for a synchronization lockfile which mod_rewrite needs to communicate with RewriteMap *programs*. Set this lockfile to a local path (not on a NFS-mounted device) when you want to use a rewriting map-program. It is not required for other types of rewriting maps.

RewriteMap

Syntax: RewriteMap *MapName MapType:MapSource*

Default: not used per default

Context: server config, virtual host

Override: *Not applicable*

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.2 (partially), Apache 1.3

The RewriteMap directive defines a *Rewriting Map* which can be used inside rule substitution strings by the mapping-functions to insert/substitute fields through a key lookup. The source of this lookup can be of various types.

The *MapName* is the name of the map and will be used to specify a mapping-function for the substitution strings of a rewriting rule via one of the following constructs:

```

 $\{$  MapName : LookupKey  $\}$ 
 $\{$  MapName : LookupKey | DefaultValue  $\}$ 

```

When such a construct occurs the map *MapName* is consulted and the key *LookupKey* is looked-up. If the key is found, the map-function construct is substituted by *SubstValue*. If the key is not found then it is substituted by *DefaultValue* or by the empty string if no *DefaultValue* was specified.

The following combinations for *MapType* and *MapSource* can be used:

- **Standard Plain Text**

MapType: txt, MapSource: Unix filesystem path to valid regular file

This is the standard rewriting map feature where the *MapSource* is a plain ASCII file containing either blank lines, comment lines (starting with a '#' character) or pairs like the following - one per line.

MatchingKey SubstValue

Example:

```

##
##  map.txt -- rewriting map
##

Ralf.S.Engelschall    rse    # Bastard Operator From Hell
Mr.Joe.Average       joe    # Mr. Average

```

```

RewriteMap real-to-user txt:/path/to/file/map.txt

```

- **Randomized Plain Text**

MapType: rnd, MapSource: Unix filesystem path to valid regular file

This is identical to the Standard Plain Text variant above but with a special post-processing feature: After looking up a value it is parsed according to contained ``|'' characters which have the meaning of ``or''. In other words they indicate a set of alternatives from which the actual returned value is chosen randomly. Although this sounds crazy and useless, it was actually designed for load balancing in a reverse proxy situation where the looked up values are server names. Example:

```

##
##  map.txt -- rewriting map
##

static    www1|www2|www3|www4
dynamic   www5|www6

```

```

RewriteMap servers rnd:/path/to/file/map.txt

```

- **Hash File**

MapType: dbm, MapSource: Unix filesystem path to valid regular file

Here the source is a binary NDBM format file containing the same contents as a *Plain Text* format file, but in a special representation which is optimized for really fast lookups. You can create such a file with any NDBM tool or with the following Perl script:

```
#!/path/to/bin/perl
##
##  txt2dbm -- convert txt map to dbm format
##

($txtmap, $dbmmap) = @ARGV;
open(TXT, "<$txtmap");
dbmopen(%DB, $dbmmap, 0644);
while (<TXT>) {
    next if (m|^s*#.*| or m|^s*$|);
    $DB{$1} = $2 if (m|^\s*(\S+)\s+(\S+)$|);
}
dbmclose(%DB);
close(TXT)
```

```
$ txt2dbm map.txt map.db
```

● Internal Function

MapType: int, MapSource: Internal Apache function

Here the source is an internal Apache function. Currently you cannot create your own, but the following functions already exists:

- **toupper:**
Converts the looked up key to all upper case.
- **tolower:**
Converts the looked up key to all lower case.
- **escape:**
Translates special characters in the looked up key to hex-encodings.
- **unescape:**
Translates hex-encodings in the looked up key back to special characters.

● External Rewriting Program

MapType: prg, MapSource: Unix filesystem path to valid regular file

Here the source is a program, not a map file. To create it you can use the language of your choice, but the result has to be a executable (*i.e.*, either object-code or a script with the magic cookie trick '#!/path/to/interpreter' as the first line).

This program is started once at startup of the Apache servers and then communicates with the rewriting engine over its `stdin` and `stdout` file-handles. For each map-function lookup it will receive the key to lookup as a newline-terminated string on `stdin`. It then has to give back the looked-up value as a newline-terminated string on `stdout` or the four-character string ```NULL``` if it fails (*i.e.*, there is no corresponding value for the given key). A trivial program which will implement a 1:1 map (*i.e.*, `key == value`) could be:

```
#!/usr/bin/perl
$_ = 1;
while (<STDIN>) {
    # ...put here any transformations or lookups...
    print $_;
}
```

But be very careful:

1. ```Keep it simple, stupid``` (KISS), because if this program hangs it will hang the Apache server when the rule occurs.
2. Avoid one common mistake: never do buffered I/O on `stdout`! This will cause a deadlock! Hence the ```$_=1``` in the above example...
3. Use the `RewriteLock` directive to define a lockfile `mod_rewrite` can use to synchronize the

communication to the program. By default no such synchronization takes place.

The RewriteMap directive can occur more than once. For each mapping-function use one RewriteMap directive to declare its rewriting mapfile. While you cannot **declare** a map in per-directory context it is of course possible to **use** this map in per-directory context.

Note: For plain text and DBM format files the looked-up keys are cached in-core until the mtime of the mapfile changes or the server does a restart. This way you can have map-functions in rules which are used for **every** request. This is no problem, because the external lookup only happens once!

RewriteBase

Syntax: RewriteBase *URL-path*

Default: *default is the physical directory path*

Context: directory, .htaccess

Override: FileInfo

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.2

The RewriteBase directive explicitly sets the base URL for per-directory rewrites. As you will see below, RewriteRule can be used in per-directory config files (.htaccess). There it will act locally, *i.e.*, the local directory prefix is stripped at this stage of processing and your rewriting rules act only on the remainder. At the end it is automatically added back to the path.

When a substitution occurs for a new URL, this module has to re-inject the URL into the server processing. To be able to do this it needs to know what the corresponding URL-prefix or URL-base is. By default this prefix is the corresponding filepath itself. **But at most websites URLs are NOT directly related to physical filename paths, so this assumption will usually be wrong!** There you have to use the RewriteBase directive to specify the correct URL-prefix.

Notice: If your webserver's URLs are **not** directly related to physical file paths, you have to use RewriteBase in every .htaccess files where you want to use RewriteRule directives.

Example:

Assume the following per-directory config file:

```
#
# /abc/def/.htaccess -- per-dir config file for directory /abc/def
# Remember: /abc/def is the physical path of /xyz, i.e., the server
#           has a 'Alias /xyz /abc/def' directive e.g.
#
RewriteEngine On

# let the server know that we were reached via /xyz and not
# via the physical path prefix /abc/def
RewriteBase /xyz

# now the rewriting rules
RewriteRule ^oldstuff\.html$ newstuff.html
```

In the above example, a request to /xyz/oldstuff.html gets correctly rewritten to the physical file /abc/def/newstuff.html.

Note - For Apache hackers:

The following list gives detailed information about the internal processing steps:

```
Request:
  /xyz/oldstuff.html

Internal Processing:
  /xyz/oldstuff.html    -> /abc/def/oldstuff.html  (per-server Alias)
  /abc/def/oldstuff.html -> /abc/def/newstuff.html  (per-dir RewriteRule)
  /abc/def/newstuff.html -> /xyz/newstuff.html    (per-dir RewriteBase)
  /xyz/newstuff.html    -> /abc/def/newstuff.html  (per-server Alias)

Result:
  /abc/def/newstuff.html
```

This seems very complicated but is the correct Apache internal processing, because the per-directory rewriting comes too late in the process. So, when it occurs the (rewritten) request has to be re-injected into the Apache kernel! BUT: While this seems like a serious overhead, it really isn't, because this re-injection happens fully internally to the Apache server and the same procedure is used by many other operations inside Apache. So, you can be sure the design and implementation is correct.

RewriteCond

Syntax: RewriteCond *TestString CondPattern*

Default: None

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.2 (partially), Apache 1.3

The RewriteCond directive defines a rule condition. Precede a RewriteRule directive with one or more RewriteCond directives. The following rewriting rule is only used if its pattern matches the current state of the URI **and** if these additional conditions apply too.

TestString is a string which can contains the following expanded constructs in addition to plain text:

- **RewriteRule backreferences:** These are backreferences of the form

\$N

(0 <= N <= 9) which provide access to the grouped parts (parenthesis!) of the pattern from the corresponding RewriteRule directive (the one following the current bunch of RewriteCond directives).

- **RewriteCond backreferences:** These are backreferences of the form

%N

(1 <= N <= 9) which provide access to the grouped parts (parentheses!) of the pattern from the last matched RewriteCond directive in the current bunch of conditions.

- **RewriteMap expansions:** These are expansions of the form

\${mapname:key|default}

See [the documentation for RewriteMap](#) for more details.

- **Server-Variables:** These are variables of the form

%{NAME_OF_VARIABLE}

where *NAME_OF_VARIABLE* can be a string taken from the following list:

HTTP headers:

HTTP_USER_AGENT
 HTTP_REFERER
 HTTP_COOKIE
 HTTP_FORWARDED
 HTTP_HOST
 HTTP_PROXY_CONNECTION
 HTTP_ACCEPT

connection & request:

REMOTE_ADDR
 REMOTE_HOST
 REMOTE_USER
 REMOTE_IDENT
 REQUEST_METHOD
 SCRIPT_FILENAME
 PATH_INFO
 QUERY_STRING
 AUTH_TYPE

server internals:

DOCUMENT_ROOT
 SERVER_ADMIN
 SERVER_NAME
 SERVER_ADDR
 SERVER_PORT
 SERVER_PROTOCOL
 SERVER_SOFTWARE

system stuff:

TIME_YEAR
 TIME_MON
 TIME_DAY
 TIME_HOUR
 TIME_MIN
 TIME_SEC
 TIME_WDAY
 TIME

specials:

API_VERSION
 THE_REQUEST
 REQUEST_URI
 REQUEST_FILENAME
 IS_SUBREQ

Notice: These variables all correspond to the similarly named HTTP MIME-headers, C variables of the Apache server or `struct tm` fields of the Unix system. Most are documented elsewhere in the Manual or in the CGI specification. Those that are special to `mod_rewrite` include:

IS_SUBREQ

Will contain the text "true" if the request currently being processed is a sub-request, "false" otherwise. Sub-requests may be generated by modules that need to resolve additional files or URIs in order to complete their tasks.

API_VERSION

This is the version of the Apache module API (the internal interface between server and module) in the current `httpd` build, as defined in `include/ap_mmn.h`. The module API version corresponds to the version of Apache in use (in the release version of Apache 1.3.14, for instance, it is 19990320:10), but is mainly of interest to module authors.

THE_REQUEST

The full HTTP request line sent by the browser to the server (e.g., "GET /index.html HTTP/1.1"). This does not include any additional headers sent by the browser.

REQUEST_URI

The resource requested in the HTTP request line. (In the example above, this would be "/index.html".)

REQUEST_FILENAME

The full local filesystem path to the file or script matching the request.

Special Notes:

1. The variables `SCRIPT_FILENAME` and `REQUEST_FILENAME` contain the same value, *i.e.*, the value of the `filename` field of the internal `request_rec` structure of the Apache server. The first name is just the commonly known CGI variable name while the second is the consistent counterpart to `REQUEST_URI` (which contains the value of the `uri` field of `request_rec`).
2. There is the special format: `%{ENV:variable}` where *variable* can be any environment variable. This is looked-up via internal Apache structures and (if not found there) via `getenv()` from the Apache server process.
3. There is the special format: `%{HTTP:header}` where *header* can be any HTTP MIME-header name. This is looked-up from the HTTP request. Example: `%{HTTP:Proxy-Connection}` is the value of the HTTP header `Proxy-Connection`.

4. There is the special format `%{LA-U:variable}` for look-aheads which perform an internal (URL-based) sub-request to determine the final value of *variable*. Use this when you want to use a variable for rewriting which is actually set later in an API phase and thus is not available at the current stage. For instance when you want to rewrite according to the `REMOTE_USER` variable from within the per-server context (`httpd.conf` file) you have to use `%{LA-U:REMOTE_USER}` because this variable is set by the authorization phases which come *after* the URL translation phase where `mod_rewrite` operates. On the other hand, because `mod_rewrite` implements its per-directory context (`.htaccess` file) via the Fixup phase of the API and because the authorization phases come *before* this phase, you just can use `%{REMOTE_USER}` there.
5. There is the special format: `%{LA-F:variable}` which performs an internal (filename-based) sub-request to determine the final value of *variable*. Most of the time this is the same as LA-U above.

CondPattern is the condition pattern, *i.e.*, a regular expression which is applied to the current instance of the *TestString*, *i.e.*, *TestString* is evaluated and then matched against *CondPattern*.

Remember: *CondPattern* is a standard *Extended Regular Expression* with some additions:

1. You can prefix the pattern string with a '!' character (exclamation mark) to specify a **non**-matching pattern.
2. There are some special variants of *CondPatterns*. Instead of real regular expression strings you can also use one of the following:
 - '<**CondPattern**>' (is lexically lower)
Treats the *CondPattern* as a plain string and compares it lexically to *TestString*. True if *TestString* is lexically lower than *CondPattern*.
 - '>**CondPattern**>' (is lexically greater)
Treats the *CondPattern* as a plain string and compares it lexically to *TestString*. True if *TestString* is lexically greater than *CondPattern*.
 - '=**CondPattern**' (is lexically equal)
Treats the *CondPattern* as a plain string and compares it lexically to *TestString*. True if *TestString* is lexically equal to *CondPattern*, *i.e.* the two strings are exactly equal (character by character). If *CondPattern* is just "" (two quotation marks) this compares *TestString* to the empty string.
 - '-**d**' (is **d**irectory)
Treats the *TestString* as a pathname and tests if it exists and is a directory.
 - '-**f**' (is regular **f**ile)
Treats the *TestString* as a pathname and tests if it exists and is a regular file.
 - '-**s**' (is regular file with **s**ize)
Treats the *TestString* as a pathname and tests if it exists and is a regular file with size greater than zero.
 - '-**l**' (is symbolic **l**ink)
Treats the *TestString* as a pathname and tests if it exists and is a symbolic link.
 - '-**F**' (is existing file via subrequest)
Checks if *TestString* is a valid file and accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to determine the check, so use it with care because it decreases your servers performance!
 - '-**U**' (is existing URL via subrequest)
Checks if *TestString* is a valid URL and accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to determine the check, so use it with care because it decreases your server's performance!

Notice: All of these tests can also be prefixed by an exclamation mark ('!') to negate their meaning.

Additionally you can set special flags for *CondPattern* by appending

`[flags]`

as the third argument to the `RewriteCond` directive. *Flags* is a comma-separated list of the following flags:

- **'nocase | NC'** (no case)
This makes the test case-insensitive, *i.e.*, there is no difference between 'A-Z' and 'a-z' both in the expanded *TestString* and the *CondPattern*. This flag is effective only for comparisons between *TestString* and *CondPattern*. It has no effect on filesystem and subrequest checks.

- **'ornext | OR'** (or next condition)

Use this to combine rule conditions with a local OR instead of the implicit AND. Typical example:

```
RewriteCond %{REMOTE_HOST} ^host1.* [OR]
RewriteCond %{REMOTE_HOST} ^host2.* [OR]
RewriteCond %{REMOTE_HOST} ^host3.*
RewriteRule ...some special stuff for any of these hosts...
```

Without this flag you would have to write the cond/rule three times.

Example:

To rewrite the Homepage of a site according to the ``User-Agent :'' header of the request, you can use the following:

```
RewriteCond %{HTTP_USER_AGENT} ^Mozilla.*
RewriteRule ^/$ /homepage.max.html [L]

RewriteCond %{HTTP_USER_AGENT} ^Lynx.*
RewriteRule ^/$ /homepage.min.html [L]

RewriteRule ^/$ /homepage.std.html [L]
```

Interpretation: If you use Netscape Navigator as your browser (which identifies itself as 'Mozilla'), then you get the max homepage, which includes Frames, *etc.* If you use the Lynx browser (which is Terminal-based), then you get the min homepage, which contains no images, no tables, *etc.* If you use any other browser you get the standard homepage.

RewriteRule

Syntax: RewriteRule *Pattern Substitution*

Default: None

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Extension

Module: mod_rewrite.c

Compatibility: Apache 1.2 (partially), Apache 1.3

The RewriteRule directive is the real rewriting workhorse. The directive can occur more than once. Each directive then defines one single rewriting rule. The **definition order** of these rules is **important**, because this order is used when applying the rules at run-time.

Pattern can be (for Apache 1.1.x a System V8 and for Apache 1.2.x and later a POSIX) regular expression which gets applied to the current URL. Here ``current'' means the value of the URL when this rule gets applied. This may not be the originally requested URL, because any number of rules may already have matched and made alterations to it.

Some hints about the syntax of regular expressions:

Text :

. Any single character
 [chars] Character class: One of chars
 [^chars] Character class: None of chars
 text1|text2 Alternative: text1 or text2

Quantifiers:

? 0 or 1 of the preceding text
 * 0 or N of the preceding text (N > 0)
 + 1 or N of the preceding text (N > 1)

Grouping:

(text) Grouping of text
 (either to set the borders of an alternative or
 for making backreferences where the **N**th group can
 be used on the RHS of a RewriteRule with **\$N**)

Anchors:

^ Start of line anchor
 \$ End of line anchor

Escaping:

\char escape that particular char
 (for instance to specify the chars ".[]()" etc.)

For more information about regular expressions either have a look at your local `regex(3)` manpage or its `src/regex/regex.3` copy in the Apache 1.3 distribution. If you are interested in more detailed information about regular expressions and their variants (POSIX regex, Perl regex, etc.) have a look at the following dedicated book on this topic:

Mastering Regular Expressions
 Jeffrey E.F. Friedl
 Nutshell Handbook Series
 O'Reilly & Associates, Inc. 1997
 ISBN 1-56592-257-3

Additionally in `mod_rewrite` the NOT character ('!') is a possible pattern prefix. This gives you the ability to negate a pattern; to say, for instance: ```if the current URL does NOT match this pattern```. This can be used for exceptional cases, where it is easier to match the negative pattern, or as a last default rule.

Notice: When using the NOT character to negate a pattern you cannot have grouped wildcard parts in the pattern. This is impossible because when the pattern does NOT match, there are no contents for the groups. In consequence, if negated patterns are used, you cannot use `$N` in the substitution string!

Substitution of a rewriting rule is the string which is substituted for (or replaces) the original URL for which *Pattern* matched. Beside plain text you can use

1. back-references `$N` to the RewriteRule pattern
2. back-references `%N` to the last matched RewriteCond pattern
3. server-variables as in rule condition test-strings (`%{VARIABLE}`)
4. [mapping-function](#) calls (`${mapname:key|default}`)

Back-references are `$N` (`N=0..9`) identifiers which will be replaced by the contents of the **N**th group of the matched *Pattern*. The server-variables are the same as for the *TestString* of a `RewriteCond` directive. The mapping-functions come from the `RewriteMap` directive and are explained there. These three types of variables are expanded in the order of the above list.

As already mentioned above, all the rewriting rules are applied to the *Substitution* (in the order of definition in the config file). The URL is **completely replaced** by the *Substitution* and the rewriting process goes on until there are no more rules unless explicitly terminated by a **L** flag - see below.

There is a special substitution string named '-' which means: **NO substitution!** Sounds silly? No, it is useful to provide rewriting rules which **only** match some URLs but do no substitution, *e.g.*, in conjunction with the **C** (chain) flag to be able to have more than one pattern to be applied before a substitution occurs.

One more note: You can even create URLs in the substitution string containing a query string part. Just use a question mark inside the substitution string to indicate that the following stuff should be re-injected into the QUERY_STRING. When you want to erase an existing query string, end the substitution string with just the question mark.

Note: There is a special feature: When you prefix a substitution field with `http://thishost[:thisport]` then **mod_rewrite** automatically strips it out. This auto-reduction on implicit external redirect URLs is a useful and important feature when used in combination with a mapping-function which generates the hostname part. Have a look at the first example in the example section below to understand this.

Remember: An unconditional external redirect to your own server will not work with the prefix `http://thishost` because of this feature. To achieve such a self-redirect, you have to use the **R**-flag (see below).

Additionally you can set special flags for *Substitution* by appending

[flags]

as the third argument to the RewriteRule directive. *Flags* is a comma-separated list of the following flags:

- **'redirect | R [=code]'** (force redirect)
Prefix *Substitution* with `http://thishost[:thisport]/` (which makes the new URL a URI) to force a external redirection. If no *code* is given a HTTP response of 302 (MOVED TEMPORARILY) is used. If you want to use other response codes in the range 300-400 just specify them as a number or use one of the following symbolic names: `temp` (default), `permanent`, `seeother`. Use it for rules which should canonicalize the URL and give it back to the client, *e.g.*, translate `~/~` into `~/u/` or always append a slash to `/u/user`, etc.
- **'forbidden | F'** (force URL to be forbidden)
This forces the current URL to be forbidden, *i.e.*, it immediately sends back a HTTP response of 403 (FORBIDDEN). Use this flag in conjunction with appropriate RewriteConds to conditionally block some URLs.
- **'gone | G'** (force URL to be gone)
This forces the current URL to be gone, *i.e.*, it immediately sends back a HTTP response of 410 (GONE). Use this flag to mark pages which no longer exist as gone.
- **'proxy | P'** (force proxy)
This flag forces the substitution part to be internally forced as a proxy request and immediately (*i.e.*, rewriting rule processing stops here) put through the [proxy module](#). You have to make sure that the substitution string is a valid URI (*e.g.*, typically starting with `http://hostname`) which can be handled by the Apache proxy module. If not you get an error from the proxy module. Use this flag to achieve a more powerful implementation of the [ProxyPass](#) directive, to map some remote stuff into the namespace of the local server.

Notice: To use this functionality make sure you have the proxy module compiled into your Apache server program. If you don't know please check whether `mod_proxy.c` is part of the ```httpd -l``` output. If yes, this functionality is available to `mod_rewrite`. If not, then you first have to rebuild the ```httpd``` program with `mod_proxy` enabled.

- **'last | L'** (last rule)
Stop the rewriting process here and don't apply any more rewriting rules. This corresponds to the Perl `last` command or the `break` command from the C language. Use this flag to prevent the currently rewritten URL from being rewritten further by following rules. For example, use it to rewrite the root-path URL (`/`) to a real one, *e.g.*, `/e/www/`.
- **'next | N'** (next round)

Re-run the rewriting process (starting again with the first rewriting rule). Here the URL to match is again not the original URL but the URL from the last rewriting rule. This corresponds to the Perl `next` command or the `continue` command from the C language. Use this flag to restart the rewriting process, *i.e.*, to immediately go to the top of the loop.

But be careful not to create an infinite loop!

- **'chain|C'** (chained with next rule)
This flag chains the current rule with the next rule (which itself can be chained with the following rule, *etc.*). This has the following effect: if a rule matches, then processing continues as usual, *i.e.*, the flag has no effect. If the rule does **not** match, then all following chained rules are skipped. For instance, use it to remove the ``.www'` part inside a per-directory rule set when you let an external redirect happen (where the ``.www'` part should not to occur!).
- **'type|T=MIME-type'** (force MIME type)
Force the MIME-type of the target file to be *MIME-type*. For instance, this can be used to simulate the `mod_alias` directive `ScriptAlias` which internally forces all files inside the mapped directory to have a MIME type of `application/x-httpd-cgi`.
- **'nosubreq|NS'** (used only if **no** internal sub-request)
This flag forces the rewriting engine to skip a rewriting rule if the current request is an internal sub-request. For instance, sub-requests occur internally in Apache when `mod_include` tries to find out information about possible directory default files (`index.xxx`). On sub-requests it is not always useful and even sometimes causes a failure to if the complete set of rules are applied. Use this flag to exclude some rules.

Use the following rule for your decision: whenever you prefix some URLs with CGI-scripts to force them to be processed by the CGI-script, the chance is high that you will run into problems (or even overhead) on sub-requests. In these cases, use this flag.

- **'nocase|NC'** (no case)
This makes the *Pattern* case-insensitive, *i.e.*, there is no difference between 'A-Z' and 'a-z' when *Pattern* is matched against the current URL.
- **'qsappend|QSA'** (query string append)
This flag forces the rewriting engine to append a query string part in the substitution string to the existing one instead of replacing it. Use this when you want to add more data to the query string via a rewrite rule.
- **'noescape|NE'** (no URI escaping of output)
This flag keeps `mod_rewrite` from applying the usual URI escaping rules to the result of a rewrite. Ordinarily, special characters (such as '%', '\$', ';', and so on) will be escaped into their hexcode equivalents ('%25', '%24', and '%3B', respectively); this flag prevents this from being done. This allows percent symbols to appear in the output, as in

```
RewriteRule /foo/(.*) /bar?arg=P1\%3d$1 [R,NE]
```

which would turn `/foo/zed` into a safe request for `/bar?arg=P1=zed`.

Notice: The `noescape` flag is only available with Apache 1.3.20 and later versions.

- **'passthrough|PT'** (pass through to next handler)
This flag forces the rewriting engine to set the `uri` field of the internal `request_rec` structure to the value of the `filename` field. This flag is just a hack to be able to post-process the output of `RewriteRule` directives by `Alias`, `ScriptAlias`, `Redirect`, *etc.* directives from other URI-to-filename translators. A trivial example to show the semantics: If you want to rewrite `/abc` to `/def` via the rewriting engine of `mod_rewrite` and then `/def` to `/ghi` with `mod_alias`:

```
RewriteRule ^/abc/(.*) /def$1 [PT]
Alias /def /ghi
```

If you omit the `PT` flag then `mod_rewrite` will do its job fine, *i.e.*, it rewrites `uri=/abc/...` to `filename=/def/...` as a full API-compliant URI-to-filename translator should do. Then `mod_alias` comes and tries to do a URI-to-filename transition which will not work.

Note: **You have to use this flag if you want to intermix directives of different modules which contain URL-to-filename translators.** The typical example is the use of `mod_alias` and `mod_rewrite`.

Note - For Apache hackers:

If the current Apache API had a filename-to-filename hook additionally to the URI-to-filename hook then we wouldn't need this flag! But without such a hook this flag is the only solution. The Apache Group has discussed this problem and will add such a hook in Apache version 2.0.

- **'skip|S=num'** (skip next rule(s))
This flag forces the rewriting engine to skip the next *num* rules in sequence when the current rule matches. Use this to make pseudo if-then-else constructs: The last rule of the then-clause becomes `skip=N` where N is the number of rules in the else-clause. (This is **not** the same as the 'chain|C' flag!)
- **'env|E=VAR:VAL'** (set environment variable)
This forces an environment variable named *VAR* to be set to the value *VAL*, where *VAL* can contain regexp backreferences `$N` and `%N` which will be expanded. You can use this flag more than once to set more than one variable. The variables can be later dereferenced in many situations, but usually from within XSSI (via `<!--#echo var="VAR" -->`) or CGI (e.g. `$ENV{ 'VAR' }`). Additionally you can dereference it in a following RewriteCond pattern via `%{ENV:VAR}`. Use this to strip but remember information from URLs.

Note: Never forget that *Pattern* is applied to a complete URL in per-server configuration files. **But in per-directory configuration files, the per-directory prefix (which always is the same for a specific directory!) is automatically removed for the pattern matching and automatically added after the substitution has been done.** This feature is essential for many sorts of rewriting, because without this prefix stripping you have to match the parent directory which is not always possible.

There is one exception: If a substitution string starts with `http://` then the directory prefix will **not** be added and an external redirect or proxy throughput (if flag **P** is used!) is forced!

Note: To enable the rewriting engine for per-directory configuration files you need to set `RewriteEngine On` in these files **and** `Options FollowSymLinks` must be enabled. If your administrator has disabled override of `FollowSymLinks` for a user's directory, then you cannot use the rewriting engine. This restriction is needed for security reasons.

Here are all possible substitution combinations and their meanings:

**Inside per-server configuration (httpd.conf)
for request `GET /somepath/pathinfo`:**

Given Rule	Resulting Substitution
<code>^/somepath(.*) otherpath\$1</code>	not supported, because invalid!
<code>^/somepath(.*) otherpath\$1 [R]</code>	not supported, because invalid!
<code>^/somepath(.*) otherpath\$1 [P]</code>	not supported, because invalid!
<code>^/somepath(.*) /otherpath\$1</code>	<code>/otherpath/pathinfo</code>
<code>^/somepath(.*) /otherpath\$1 [R]</code>	<code>http://thishost/otherpath/pathinfo</code> via external redirection
<code>^/somepath(.*) /otherpath\$1 [P]</code>	not supported, because silly!
<code>^/somepath(.*) http://thishost/otherpath\$1</code>	<code>/otherpath/pathinfo</code>
<code>^/somepath(.*) http://thishost/otherpath\$1 [R]</code>	<code>http://thishost/otherpath/pathinfo</code> via external redirection
<code>^/somepath(.*) http://thishost/otherpath\$1 [P]</code>	not supported, because silly!

```

-----
^/somepath(.*) http://otherhost/otherpath$1 http://otherhost/otherpath/pathinfo
via external redirection

^/somepath(.*) http://otherhost/otherpath$1 [R] http://otherhost/otherpath/pathinfo
via external redirection
(the [R] flag is redundant)

^/somepath(.*) http://otherhost/otherpath$1 [P] http://otherhost/otherpath/pathinfo
via internal proxy

```

Inside per-directory configuration for /somepath

(i.e., file `.htaccess` in dir `/physical/path/to/somepath` containing `RewriteBase /somepath`)
for request `GET /somepath/localpath/pathinfo`:

Given Rule	Resulting Substitution
----- ^localpath(.*) otherpath\$1	----- /somepath/otherpath/pathinfo
^localpath(.*) otherpath\$1 [R] http://thishost/somepath/otherpath/pathinfo	via external redirection
^localpath(.*) otherpath\$1 [P]	not supported, because silly!
----- ^localpath(.*) /otherpath\$1	----- /otherpath/pathinfo
^localpath(.*) /otherpath\$1 [R]	http://thishost/otherpath/pathinfo via external redirection
^localpath(.*) /otherpath\$1 [P]	not supported, because silly!
----- ^localpath(.*) http://thishost/otherpath\$1	----- /otherpath/pathinfo
^localpath(.*) http://thishost/otherpath\$1 [R]	http://thishost/otherpath/pathinfo via external redirection
^localpath(.*) http://thishost/otherpath\$1 [P]	not supported, because silly!
----- ^localpath(.*) http://otherhost/otherpath\$1	----- http://otherhost/otherpath/pathinfo via external redirection
^localpath(.*) http://otherhost/otherpath\$1 [R]	http://otherhost/otherpath/pathinfo via external redirection (the [R] flag is redundant)
^localpath(.*) http://otherhost/otherpath\$1 [P]	http://otherhost/otherpath/pathinfo via internal proxy

Example:

We want to rewrite URLs of the form

/ Language /~ Realname / . . . / File

into

/u/ Username / . . . / File . Language

We take the rewrite mapfile from above and save it under `/path/to/file/map.txt`. Then we only have to add the following lines to the Apache server configuration file:

```

RewriteLog    /path/to/file/rewrite.log
RewriteMap    real-to-user          txt:/path/to/file/map.txt

```

```
RewriteRule ^/([\^/]+)/~([\^/]+)/(.*)$ /u/${real-to-user:$2|nobody}/$3.$1
```

Miscellaneous

Environment Variables

This module keeps track of two additional (non-standard) CGI/SSI environment variables named `SCRIPT_URL` and `SCRIPT_URI`. These contain the *logical* Web-view to the current resource, while the standard CGI/SSI variables `SCRIPT_NAME` and `SCRIPT_FILENAME` contain the *physical* System-view.

Notice: These variables hold the URI/URL *as they were initially requested, i.e., before* any rewriting. This is important because the rewriting process is primarily used to rewrite logical URLs to physical pathnames.

Example:

```
SCRIPT_NAME=/sw/lib/w3s/tree/global/u/rse/.www/index.html
SCRIPT_FILENAME=/u/rse/.www/index.html
SCRIPT_URL=/u/rse/
SCRIPT_URI=http://en1.engelschall.com/u/rse/
```

Practical Solutions

We also have an [URL Rewriting Guide](#) available, which provides a collection of practical solutions for URL-based problems. There you can find real-life rulesets and additional information about `mod_rewrite`.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_setenvif

This module provides the ability to set environment variables based upon attributes of the request.

Status: Base

Source File: mod_setenvif.c

Module Identifier: setenvif_module

Compatibility: Available in Apache 1.3 and later.

Summary

The mod_setenvif module allows you to set environment variables according to whether different aspects of the request match [regular expressions](#) you specify. These environment variables can be used by other parts of the server to make decisions about actions to be taken.

The directives are considered in the order they appear in the configuration files. So more complex sequences can be used, such as this example, which sets `netscape` if the browser is mozilla but not MSIE.

```
BrowserMatch ^Mozilla netscape
BrowserMatch MSIE !netscape
```

For additional information, we provide a document on [Environment Variables in Apache](#).

Directives

- [BrowserMatch](#)
- [BrowserMatchNoCase](#)
- [SetEnvIf](#)
- [SetEnvIfNoCase](#)

BrowserMatch directive

Syntax: BrowserMatch *regex env-variable*[=*value*] [*env-variable*[=*value*]] ...

Default: none

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_setenvif

Compatibility: Apache 1.2 and above (in Apache 1.2 this directive was found in the now-obsolete mod_browser module); use in .htaccess files only supported with 1.3.13 and later

The BrowserMatch directive defines environment variables based on the User-Agent HTTP request header field. The first argument should be a POSIX.2 extended regular expression (similar to an egrep-style regex). The rest of the arguments give the names of variables to set, and optionally values to which they should be set. These take the form of

1. *varname*, or
2. *!varname*, or
3. *varname=value*

In the first form, the value will be set to "1". The second will remove the given variable if already defined, and the third will set the variable to the value given by *value*. If a User-Agent string matches more than one entry, they will be merged. Entries are processed in the order in which they appear, and later entries can override earlier ones.

For example:

```
BrowserMatch ^Mozilla forms jpeg=yes browser=netscape
BrowserMatch "^Mozilla/[2-3]" tables agif frames javascript
BrowserMatch MSIE !javascript
```

Note that the regular expression string is **case-sensitive**. For case-INsensitive matching, see the [BrowserMatchNoCase](#) directive.

The BrowserMatch and BrowserMatchNoCase directives are special cases of the [SetEnvIf](#) and [SetEnvIfNoCase](#) directives. The following two lines have the same effect:

```
BrowserMatchNoCase Robot is_a_robot
SetEnvIfNoCase User-Agent Robot is_a_robot
```

BrowserMatchNoCase directive

Syntax: BrowserMatchNoCase *regex env-variable[=value] [env-variable[=value]] ...*

Default: none

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_setenvif

Compatibility: Apache 1.2 and above (in Apache 1.2 this directive was found in the now-obsolete mod_browser module)

The BrowserMatchNoCase directive is semantically identical to the [BrowserMatch](#) directive. However, it provides for case-insensitive matching. For example:

```
BrowserMatchNoCase mac platform=macintosh
BrowserMatchNoCase win platform=windows
```

The BrowserMatch and BrowserMatchNoCase directives are special cases of the [SetEnvIf](#) and [SetEnvIfNoCase](#) directives. The following two lines have the same effect:

```
BrowserMatchNoCase Robot is_a_robot
SetEnvIfNoCase User-Agent Robot is_a_robot
```

SetEnvIf directive

Syntax: SetEnvIf *attribute regex env-variable*[=*value*] [*env-variable*[=*value*]] ...

Default: none

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_setenvif

Compatibility: Apache 1.3 and above; the Request_Protocol keyword and environment-variable matching are only available with 1.3.7 and later; use in .htaccess files only supported with 1.3.13 and later

The SetEnvIf directive defines environment variables based on attributes of the request. These attributes can be the values of various HTTP request header fields (see [RFC2616](#) for more information about these), or of other aspects of the request, including the following:

- Remote_Host - the hostname (if available) of the client making the request
- Remote_Addr - the IP address of the client making the request
- Remote_User - the authenticated username (if available)
- Request_Method - the name of the method being used (GET, POST, *et cetera*)
- Request_Protocol - the name and version of the protocol with which the request was made (*e.g.*, "HTTP/0.9", "HTTP/1.1", *etc.*)
- Request_URI - the portion of the URL following the scheme and host portion

Some of the more commonly used request header field names include Host, User-Agent, and Referer.

If the *attribute* name doesn't match any of the special keywords, nor any of the request's header field names, it is tested as the name of an environment variable in the list of those associated with the request. This allows SetEnvIf directives to test against the result of prior matches.

Only those environment variables defined by earlier SetEnvIf[NoCase] directives are available for testing in this manner. 'Earlier' means that they were defined at a broader scope (such as server-wide) or previously in the current directive's scope.

Example:

```

SetEnvIf Request_URI "\.gif$" object_is_image=gif
SetEnvIf Request_URI "\.jpg$" object_is_image=jpg
SetEnvIf Request_URI "\.xpm$" object_is_image=xpm
:
SetEnvIf Referer www\.\mydomain\.\com intra_site_referral
:
SetEnvIf object_is_image xpm XBIT_PROCESSING=1

```

The first three will set the environment variable `object_is_image` if the request was for an image file, and the fourth sets `intra_site_referral` if the referring page was somewhere on the `www.mydomain.com` Web site.

SetEnvIfNoCase directive

Syntax: SetEnvIfNoCase *attribute regex env-variable*[=*value*] [*env-variable*[=*value*]] ...

Default: none

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: Base

Module: mod_setenvif

Compatibility: Apache 1.3 and above; the Request_Protocol keyword and environment-variable matching are only available with 1.3.7 and later; use in .htaccess files only supported with 1.3.13 and later

The SetEnvIfNoCase is semantically identical to the [SetEnvIf](#) directive, and differs only in that the regular expression matching is performed in a case-insensitive manner. For example:

```
SetEnvIfNoCase Host Apache\.Org site=apache
```

This will cause the site environment variable to be set to "apache" if the HTTP request header field Host: was included and contained Apache.Org, apache.org, or any other combination.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_so

This module provides for loading of executable code and modules into the server at start-up or restart time.

Status: Base (Windows); Experimental (Unix)

Source File: mod_so.c

Module Identifier: so_module

Compatibility: Available in Apache 1.3 and later.

Summary

This is an experimental module. On selected operating systems it can be used to load modules into Apache at runtime via the [Dynamic Shared Object](#) (DSO) mechanism, rather than requiring a recompilation.

On Unix, the loaded code typically comes from shared object files (usually with .so extension), whilst on Windows this module loads DLL files. This module is only available in Apache 1.3 and up.

In previous releases, the functionality of this module was provided for Unix by mod_dld, and for Windows by mod_dll. On Windows, mod_dll was used in beta release 1.3b1 through 1.3b5. mod_so combines these two modules into a single module for all operating systems.

Directives

- [LoadFile](#)
- [LoadModule](#)

Creating DLL Modules for Windows

The Apache module API is unchanged between the Unix and Windows versions. Many modules will run on Windows with no or little change from Unix, although others rely on aspects of the Unix architecture which are not present in Windows, and will not work.

When a module does work, it can be added to the server in one of two ways. As with Unix, it can be compiled into the server. Because Apache for Windows does not have the `Configure` program of Apache for Unix, the module's source file must be added to the ApacheCore project file, and its symbols must be added to the `os\win32\modules.c` file.

The second way is to compile the module as a DLL, a shared library that can be loaded into the server at runtime, using the [LoadModule](#) directive. These module DLLs can be distributed and run on any Apache for Windows installation, without recompilation of the server.

To create a module DLL, a small change is necessary to the module's source file: The module record must be exported from the DLL (which will be created later; see below). To do this, add the `MODULE_VAR_EXPORT` (defined in the Apache header files) to your module's module record definition. For example, if your module has:

```
module foo_module;
```

Replace the above with:

```
module MODULE_VAR_EXPORT foo_module;
```

Note that this will only be activated on Windows, so the module can continue to be used, unchanged, with Unix if needed. Also, if you are familiar with .DEF files, you can export the module record with that method instead.

Now, create a DLL containing your module. You will need to link this against the ApacheCore.lib export library that is created when the ApacheCore.dll shared library is compiled. You may also have to change the compiler settings to ensure that the Apache header files are correctly located.

This should create a DLL version of your module. Now simply place it in the modules directory of your server root, and use the [LoadModule](#) directive to load it.

LoadFile directive

Syntax: `LoadFile filename [filename] ...`

Context: server config

Status: Base

Module: mod_so

The LoadFile directive links in the named object files or libraries when the server is started or restarted; this is used to load additional code which may be required for some module to work. *Filename* is either an absolute path or relative to [ServerRoot](#).

LoadModule directive

Syntax: `LoadModule module filename`

Context: server config

Status: Base

Module: mod_so

The LoadModule directive links in the object file or library *filename* and adds the module structure named *module* to the list of active modules. *Module* is the name of the external variable of type `module` in the file, and is listed as the [Module Identifier](#) in the module documentation. Example (Unix, and for Windows as of Apache 1.3.15):

```
LoadModule status_module modules/mod_status.so
```

Example (Windows prior to Apache 1.3.15, and some 3rd party modules):

```
LoadModule foo_module modules/ApacheModuleFoo.dll
```

Note that all modules bundled with the Apache Win32 binary distribution were renamed as of Apache version 1.3.15.

Win32 Apache modules are often distributed with the old style names, or even a name such as libfoo.dll. Whatever the name of the module, the LoadModule directive requires the exact filename, no assumption is made about the filename extension.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_speling

This module attempts to correct misspellings of URLs that users might have entered, by ignoring capitalization and by allowing up to one misspelling.

Status: Extension

Source File: mod_speling.c

Module Identifier: speling_module

Compatibility: Available in Apache 1.3 and later. Available as an External module in Apache 1.1 and later.

Summary

Requests to documents sometimes cannot be served by the core apache server because the request was misspelled or miscapitalized. This module addresses this problem by trying to find a matching document, even after all other modules gave up. It does its work by comparing each document name in the requested directory against the requested document name **without regard to case**, and allowing **up to one misspelling** (character insertion / omission / transposition or wrong character). A list is built with all document names which were matched using this strategy.

If, after scanning the directory,

- no matching document was found, Apache will proceed as usual and return a "document not found" error.
- only one document is found that "almost" matches the request, then it is returned in the form of a redirection response.
- more than one document with a close match was found, then the list of the matches is returned to the client, and the client can select the correct candidate.

Directives

- [CheckSpelling](#)

CheckSpelling directive

Syntax: CheckSpelling on|off

Default: CheckSpelling Off

Context: server config, virtual host, directory, .htaccess

Override: Options

Status: Base

Module: mod_speling

Compatibility: CheckSpelling was available as a separately available module for Apache 1.1, but was limited to miscapitalizations.

As of Apache 1.3, it is part of the Apache distribution. Prior to Apache 1.3.2, the CheckSpelling directive was only available in the "server" and "virtual host" contexts.

This directive enables or disables the spelling module. When enabled, keep in mind that

- the directory scan which is necessary for the spelling correction will have an impact on the server's performance when many spelling corrections have to be performed at the same time.

- the document trees should not contain sensitive files which could be matched inadvertently by a spelling "correction".
 - the module is unable to correct misspelled user names (as in `http://my.host/~apahce/`), just file names or directory names.
 - spelling corrections apply strictly to existing files, so a request for the `<Location /status>` may get incorrectly treated as the negotiated file `"/stats.html"`.
-

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_status

This module provides information on server activity and performance.

Status: Base

Source File: mod_status.c

Module Identifier: status_module

Compatibility: Available in Apache 1.1 and later.

Summary

The Status module allows a server administrator to find out how well their server is performing. A HTML page is presented that gives the current server statistics in an easily readable form. If required this page can be made to automatically refresh (given a compatible browser). Another page gives a simple machine-readable list of the current server state.

The details given are:

- The number of children serving requests
- The number of idle children
- The status of each child, the number of requests that child has performed and the total number of bytes served by the child (*)
- A total number of accesses and byte count served (*)
- The time the server was started/restarted and the time it has been running for
- Averages giving the number of requests per second, the number of bytes served per second and the average number of bytes per request (*)
- The current percentage CPU used by each child and in total by Apache (*)
- The current hosts and requests being processed (*)

Details marked "(*)" are only available with `ExtendedStatus On`.

Directives

- [ExtendedStatus](#)

Enabling Status Support

To enable status reports only for browsers from the foo.com domain add this code to your `httpd.conf` configuration file

```
<Location /server-status>
  SetHandler server-status

  Order Deny,Allow
  Deny from all
  Allow from .foo.com
```

```
</Location>
```

You can now access server statistics by using a Web browser to access the page `http://your.server.name/server-status`

Note that mod_status will only work when you are running Apache in [standalone](#) mode and not [inetd](#) mode.

Automatic Updates

You can get the status page to update itself automatically if you have a browser that supports "refresh". Access the page `http://your.server.name/server-status?refresh=N` to refresh the page every N seconds.

Machine Readable Status File

A machine-readable version of the status file is available by accessing the page `http://your.server.name/server-status?auto`. This is useful when automatically run, see the Perl program in the /support directory of Apache, `log_server_status`.

It should be noted that if mod_status is compiled into the server, its handler capability is available in *all* configuration files, including *per*-directory files (e.g., `.htaccess`). This may have security-related ramifications for your site.

ExtendedStatus directive

Syntax: ExtendedStatus On|Off

Default: ExtendedStatus Off

Context: server config

Status: Base

Module: mod_status

Compatibility: ExtendedStatus is only available in Apache 1.3.2 and later.

This directive controls whether the server keeps track of extended status information for each request. This is only useful if the status module is enabled on the server.

This setting applies to the entire server, and cannot be enabled or disabled on a virtualhost-by-virtualhost basis.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_unique_id

This module provides an environment variable with a unique identifier for each request.

Status: Extension

Source File: mod_unique_id.c

Module Identifier: unique_id_module

Compatibility: Available in Apache 1.3 and later.

Summary

This module provides a magic token for each request which is guaranteed to be unique across "all" requests under very specific conditions. The unique identifier is even unique across multiple machines in a properly configured cluster of machines. The environment variable `UNIQUE_ID` is set to the identifier for each request. Unique identifiers are useful for various reasons which are beyond the scope of this document.

Directives

This module has no directives.

Theory

First a brief recap of how the Apache server works on Unix machines. On Unix machines, Apache creates several children, the children process requests one at a time. Each child can serve multiple requests in its lifetime. For the purpose of this discussion, the children don't share any data with each other. We'll refer to the children as httpd processes.

Your website has one or more machines under your administrative control, together we'll call them a cluster of machines. Each machine can possibly run multiple instances of Apache. All of these collectively are considered "the universe", and with certain assumptions we'll show that in this universe we can generate unique identifiers for each request, without extensive communication between machines in the cluster.

The machines in your cluster should satisfy these requirements. (Even if you have only one machine you should synchronize its clock with NTP.)

- The machines' times are synchronized via NTP or other network time protocol.
- The machines' hostnames all differ, such that the module can do a hostname lookup on the hostname and receive a different IP address for each machine in the cluster.

As far as operating system assumptions go, we assume that pids (process ids) fit in 32-bits. If the operating system uses more than 32-bits for a pid, the fix is trivial but must be performed in the code.

Given those assumptions, at a single point in time we can identify any httpd process on any machine in the cluster from all other httpd processes. The machine's IP address and the pid of the httpd process are sufficient to do this. So in order to generate unique identifiers for requests we need only distinguish between different points in time.

To distinguish time we will use a Unix timestamp (seconds since January 1, 1970 UTC), and a 16-bit counter. The timestamp has only one second granularity, so the counter is used to represent up to 65536 values during a single second. The quadruple (*ip_addr*, *pid*, *time_stamp*, *counter*) is sufficient to enumerate 65536 requests per second per httpd process. There are issues however with pid

reuse over time, and the counter is used to alleviate this issue.

When an httpd child is created, the counter is initialized with (current microseconds divided by 10) modulo 65536 (this formula was chosen to eliminate some variance problems with the low order bits of the microsecond timers on some systems). When a unique identifier is generated, the time stamp used is the time the request arrived at the web server. The counter is incremented every time an identifier is generated (and allowed to roll over).

The kernel generates a pid for each process as it forks the process, and pids are allowed to roll over (they're 16-bits on many Unixes, but newer systems have expanded to 32-bits). So over time the same pid will be reused. However unless it is reused within the same second, it does not destroy the uniqueness of our quadruple. That is, we assume the system does not spawn 65536 processes in a one second interval (it may even be 32768 processes on some Unixes, but even this isn't likely to happen).

Suppose that time repeats itself for some reason. That is, suppose that the system's clock is screwed up and it revisits a past time (or it is too far forward, is reset correctly, and then revisits the future time). In this case we can easily show that we can get pid and time stamp reuse. The choice of initializer for the counter is intended to help defeat this. Note that we really want a random number to initialize the counter, but there aren't any readily available numbers on most systems (*i.e.*, you can't use rand() because you need to seed the generator, and can't seed it with the time because time, at least at one second resolution, has repeated itself). This is not a perfect defense.

How good a defense is it? Suppose that one of your machines serves at most 500 requests per second (which is a very reasonable upper bound at this writing, because systems generally do more than just shovel out static files). To do that it will require a number of children which depends on how many concurrent clients you have. But we'll be pessimistic and suppose that a single child is able to serve 500 requests per second. There are 1000 possible starting counter values such that two sequences of 500 requests overlap. So there is a 1.5% chance that if time (at one second resolution) repeats itself this child will repeat a counter value, and uniqueness will be broken. This was a very pessimistic example, and with real world values it's even less likely to occur. If your system is such that it's still likely to occur, then perhaps you should make the counter 32 bits (by editing the code).

You may be concerned about the clock being "set back" during summer daylight savings. However this isn't an issue because the times used here are UTC, which "always" go forward. Note that x86 based Unixes may need proper configuration for this to be true -- they should be configured to assume that the motherboard clock is on UTC and compensate appropriately. But even still, if you're running NTP then your UTC time will be correct very shortly after reboot.

The UNIQUE_ID environment variable is constructed by encoding the 112-bit (32-bit IP address, 32 bit pid, 32 bit time stamp, 16 bit counter) quadruple using the alphabet [A-Za-z0-9@-] in a manner similar to MIME base64 encoding, producing 19 characters. The MIME base64 alphabet is actually [A-Za-z0-9+/] however + and / need to be specially encoded in URLs, which makes them less desirable. All values are encoded in network byte ordering so that the encoding is comparable across architectures of different byte ordering. The actual ordering of the encoding is: time stamp, IP address, pid, counter. This ordering has a purpose, but it should be emphasized that applications should not dissect the encoding. Applications should treat the entire encoded UNIQUE_ID as an opaque token, which can be compared against other UNIQUE_IDs for equality only.

The ordering was chosen such that it's possible to change the encoding in the future without worrying about collision with an existing database of UNIQUE_IDs. The new encodings should also keep the time stamp as the first element, and can otherwise use the same alphabet and bit length. Since the time stamps are essentially an increasing sequence, it's sufficient to have a *flag second* in which all machines in the cluster stop serving and request, and stop using the old encoding format. Afterwards they can resume requests and begin issuing the new encodings.

This is a relatively portable solution. It is extended to multithreaded systems like Windows NT, which add the thread-id to the ID, producing a 144-bit (including 32-bit tid) quadruple that generates a 24 character UNIQUE_ID value. The identifiers generated have essentially an infinite life-time because future identifiers can be made longer as required. Essentially no communication is required between machines in the cluster (only NTP synchronization is required, which is low overhead), and no communication between httpd processes is required (the communication is implicit in the pid value assigned by the kernel). In very specific situations the identifier can be shortened, but more information needs to be assumed (for example the 32-bit IP address is overkill for any site, but there is no portable shorter replacement for it). This module may be extended to include an entire IPv6 address, but that is overkill for nearly all server configurations.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_userdir

This module provides for user-specific directories.

Status: Base

Source File: mod_userdir.c

Module Identifier: userdir_module

Directives

- [UserDir](#)

UserDir directive

Syntax: UserDir *directory-filename*

Default: UserDir public_html

Context: server config, virtual host

Status: Base

Module: mod_userdir

Compatibility: All forms except the UserDir public_html form are only available in Apache 1.1 or above. Use of the enabled keyword, or disabled with a list of usernames, is only available in Apache 1.3 and above.

The UserDir directive sets the real directory in a user's home directory to use when a request for a document for a user is received. *Directory-filename* is one of the following:

- The name of a directory or a pattern such as those shown below.
- The keyword disabled. This turns off *all* username-to-directory translations except those explicitly named with the enabled keyword (see below).
- The keyword disabled followed by a space-delimited list of usernames. Usernames that appear in such a list will *never* have directory translation performed, even if they appear in an enabled clause.
- The keyword enabled followed by a space-delimited list of usernames. These usernames will have directory translation performed even if a global disable is in effect, but not if they also appear in a disabled clause.

If neither the enabled nor the disabled keywords appear in the Userdir directive, the argument is treated as a filename pattern, and is used to turn the name into a directory specification. A request for `http://www.foo.com/~bob/one/two.html` will be translated to:

```
UserDir public_html      -> ~bob/public_html/one/two.html
UserDir /usr/web         -> /usr/web/bob/one/two.html
UserDir /home/*/www      -> /home/bob/www/one/two.html
```

The following directives will send redirects to the client:

```
UserDir http://www.foo.com/users -> http://www.foo.com/users/bob/one/two.html
UserDir http://www.foo.com/*/usr -> http://www.foo.com/bob/usr/one/two.html
```

UserDir http://www.foo.com/~*/ -> http://www.foo.com/~bob/one/two.html

Be careful when using this directive; for instance, "UserDir ./" would map "~/root" to "/" - which is probably undesirable. If you are running Apache 1.3 or above, it is strongly recommended that your configuration include a "UserDir disabled root" declaration. See also the [<Directory>](#) directive and the [Security Tips](#) page for more information.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_usertrack

This module uses cookies to provide for a *clickstream* log of user activity on a site.

Status: Extension

Source File: mod_usertrack.c

Module Identifier: usertrack_module

Compatibility: Known as mod_cookies prior to Apache 1.3.

Summary

Previous releases of Apache have included a module which generates a 'clickstream' log of user activity on a site using cookies. This was called the "cookies" module, mod_cookies. In Apache 1.2 and later this module has been renamed the "user tracking" module, mod_usertrack. This module has been simplified and new directives added.

Directives

- [CookieDomain](#)
- [CookieExpires](#)
- [CookieName](#)
- [CookieStyle](#)
- [CookieTracking](#)

Logging

Previously, the cookies module (now the user tracking module) did its own logging, using the `CookieLog` directive. In this release, this module does no logging at all. Instead, a configurable log format file should be used to log user click-streams. This is possible because the logging module now allows [multiple log files](#). The cookie itself is logged by using the text `%{cookie}n` in the log file format. For example:

```
CustomLog logs/clickstream "%{cookie}n %r %t"
```

For backward compatibility the configurable log module implements the old `CookieLog` directive, but this should be upgraded to the above `CustomLog` directive.

2-digit or 4-digit dates for cookies?

(the following is from message <022701bda43d\$9d32bbb0\$1201a8c0@christian.office.sane.com> in the new-httpd archives)

```
From: "Christian Allen" <christian@sane.com>  
Subject: Re: Apache Y2K bug in mod_usertrack.c  
Date: Tue, 30 Jun 1998 11:41:56 -0400
```

Did some work with cookies and dug up some info that might be useful.

True, Netscape claims that the correct format NOW is four digit dates, and four digit dates do in fact work... for Netscape 4.x (Communicator), that is. However, 3.x and below do NOT accept them. It seems that Netscape originally had a 2-digit standard, and then with all of the Y2K hype and probably a few complaints, changed to a four digit date for Communicator. Fortunately, 4.x also understands the 2-digit format, and so the best way to ensure that your expiration date is legible to the client's browser is to use 2-digit dates.

However, this does not limit expiration dates to the year 2000; if you use an expiration year of "13", for example, it is interpreted as 2013, NOT 1913! In fact, you can use an expiration year of up to "37", and it will be understood as "2037" by both MSIE and Netscape versions 3.x and up (not sure about versions previous to those). Not sure why Netscape used that particular year as its cut-off point, but my guess is that it was in respect to UNIX's 2038 problem. Netscape/MSIE 4.x seem to be able to understand 2-digit years beyond that, at least until "50" for sure (I think they understand up until about "70", but not for sure).

Summary: Mozilla 3.x and up understands two digit dates up until "37" (2037). Mozilla 4.x understands up until at least "50" (2050) in 2-digit form, but also understands 4-digit years, which can probably reach up until 9999. Your best bet for sending a long-life cookie is to send it for some time late in the year "37".

CookieDomain directive

Syntax: CookieDomain *domain*

Context: server config, virtual host, directory, .htaccess

Status: optional

Module: mod_usertrack **Compatibility:** Apache 1.3.21 and later

This directive controls the setting of the domain to which the tracking cookie applies. If not present, no domain is included in the cookie header field.

The domain string **must** begin with a dot, and **must** include at least one embedded dot. That is, ".foo.com" is legal, but "foo.bar.com" and ".com" are not.

CookieExpires directive

Syntax: CookieExpires *expiry-period*

Context: **1.3.20 and earlier:** server config, virtual host; **1.3.21 and later:** server config, virtual host, directory, .htaccess

Status: optional

Module: mod_usertrack

When used, this directive sets an expiry time on the cookie generated by the usertrack module. The *expiry-period* can be given either as a number of seconds, or in the format such as "2 weeks 3 days 7 hours". Valid denominations are: years, months, weeks, hours, minutes and seconds. If the expiry time is in any format other than one number indicating the number of seconds, it must be enclosed by double quotes.

If this directive is not used, cookies last only for the current browser session.

CookieName directive

Syntax: CookieName *token*

Default: *Apache*

Context: server config, virtual host, directory, .htaccess

Status: optional

Module: mod_usertrack

Compatibility: Apache 1.3.7 and later

This directive allows you to change the name of the cookie this module uses for its tracking purposes. By default the cookie is named "Apache".

You must specify a valid cookie name; results are unpredictable if you use a name containing unusual characters. Valid characters include A-Z, a-z, 0-9, "_", and "-".

CookieStyle directive

Syntax: CookieStyle *Netscape/Cookie/Cookie2/RFC2109/RFC2965*

Context: server config, virtual host, directory, .htaccess

Status: optional

Module: mod_usertrack

This directive controls the format of the cookie header field. The three formats allowed are:

- **Netscape**, which is the original but now deprecated syntax. This is the default, and the syntax Apache has historically used.
- **Cookie** or **RFC2109**, which is the syntax that superseded the Netscape syntax.
- **Cookie2** or **RFC2965**, which is the most current cookie syntax.

Not all clients can understand all of these formats. but you should use the newest one that is generally acceptable to your users' browsers.

CookieTracking directive

Syntax: CookieTracking on|off

Context: server config, virtual host, directory, .htaccess

Override: FileInfo

Status: optional

Module: mod_usertrack

When the user track module is compiled in, and "CookieTracking on" is set, Apache will start sending a user-tracking cookie for all new requests. This directive can be used to turn this behavior on or off on a per-server or per-directory basis. By default, compiling mod_usertrack will not activate cookies.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Module mod_vhost_alias

This module provides support for [dynamically configured mass virtual hosting](#).

Status: Extension

Source File: mod_vhost_alias.c

Module Identifier: vhost_alias_module

Compatibility: Available in Apache 1.3.7 and later.

Summary

This module creates dynamically configured virtual hosts, by allowing the IP address and/or the `Host` : header of the HTTP request to be used as part of the pathname to determine what files to serve. This allows for easy use of a huge number of virtual hosts with similar configurations.

Directives

- [VirtualDocumentRoot](#)
- [VirtualDocumentRootIP](#)
- [VirtualScriptAlias](#)
- [VirtualScriptAliasIP](#)

See also: [UseCanonicalName](#).

Directory Name Interpolation

All the directives in this module interpolate a string into a pathname. The interpolated string (henceforth called the "name") may be either the server name (see the [UseCanonicalName](#) directive for details on how this is determined) or the IP address of the virtual host on the server in dotted-quad format. The interpolation is controlled by specifiers inspired by `printf` which have a number of formats:

`%%`

insert a %

`%P`

insert the port number of the virtual host

`%N.M`

insert (part of) the name

`N` and `M` are used to specify substrings of the name. `N` selects from the dot-separated components of the name, and `M` selects characters within whatever `N` has selected. `M` is optional and defaults to zero if it isn't present; the dot must be present if and only if `M` is present. The interpretation is as follows:

0

Apache module mod_vhost_alias

the whole name

1

the first part

2

the second part

-1

the last part

-2

the penultimate part

2+

the second and all subsequent parts

-2+

the penultimate and all preceding parts

1+ and -1+

the same as 0

If N or M is greater than the number of parts available a single underscore is interpolated.

Examples

For simple name-based virtual hosts you might use the following directives in your server configuration file:

```
UseCanonicalName    Off
VirtualDocumentRoot /usr/local/apache/vhosts/%0
```

A request for `http://www.example.com/directory/file.html` will be satisfied by the file `/usr/local/apache/vhosts/www.example.com/directory/file.html`.

For a very large number of virtual hosts it is a good idea to arrange the files to reduce the size of the `vhosts` directory. To do this you might use the following in your configuration file:

```
UseCanonicalName    Off
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.1/%2.2/%2.3/%2
```

A request for `http://www.example.isp.com/directory/file.html` will be satisfied by the file `/usr/local/apache/vhosts/isp.com/e/x/a/example/directory/file.html`. A more even spread of files can be achieved by hashing from the end of the name, for example:

```
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.-1/%2.-2/%2.-3/%2
```

The example request would come from `/usr/local/apache/vhosts/isp.com/e/l/p/example/directory/file.html`. Alternatively you might use:

```
VirtualDocumentRoot /usr/local/apache/vhosts/%3+/%2.1/%2.2/%2.3/%2.4+
```

The example request would come from `/usr/local/apache/vhosts/isp.com/e/x/a/mple/directory/file.html`.

For IP-based virtual hosting you might use the following in your configuration file:

```
UseCanonicalName    DNS
VirtualDocumentRootIP /usr/local/apache/vhosts/%1/%2/%3/%4/docs
VirtualScriptAliasIP /usr/local/apache/vhosts/%1/%2/%3/%4/cgi-bin
```

A request for `http://www.example.isp.com/directory/file.html` would be satisfied by the file

/usr/local/apache/vhosts/10/20/30/40/docs/directory/file.html if the IP address of www.example.com were 10.20.30.40. A request for http://www.example.com/cgi-bin/script.pl would be satisfied by executing the program /usr/local/apache/vhosts/10/20/30/40/cgi-bin/script.pl.

If you want to include the . character in a VirtualDocumentRoot directive, but it clashes with a % directive, you can work around the problem in the following way:

```
VirtualDocumentRoot /usr/local/apache/vhosts/%2.0.%3.0
```

A request for http://www.example.com/directory/file.html will be satisfied by the file /usr/local/apache/vhosts/example.com/directory/file.html.

The [LogFormat directives](#) %V and %A are useful in conjunction with this module.

VirtualDocumentRoot directive

Syntax: VirtualDocumentRoot *interpolated-directory*

Default: None

Context: server config, virtual host

Status: Extension

Module: mod_vhost_alias

Compatibility: VirtualDocumentRoot is only available in 1.3.7 and later.

The VirtualDocumentRoot directive allows you to determine where Apache will find your documents based on the value of the server name. The result of expanding *interpolated-directory* is used as the root of the document tree in a similar manner to the [DocumentRoot](#) directive's argument. If *interpolated-directory* is none then VirtualDocumentRoot is turned off. This directive cannot be used in the same context as [VirtualDocumentRootIP](#).

VirtualDocumentRootIP directive

Syntax: VirtualDocumentRootIP *interpolated-directory*

Default: None

Context: server config, virtual host

Status: Extension

Module: mod_vhost_alias

Compatibility: VirtualDocumentRootIP is only available in 1.3.7 and later.

The VirtualDocumentRootIP directive is like the [VirtualDocumentRoot](#) directive, except that it uses the IP address of the server end of the connection instead of the server name.

VirtualScriptAlias directive

Syntax: VirtualScriptAlias *interpolated-directory*

Default: None

Context: server config, virtual host

Status: Extension

Module: mod_vhost_alias

Compatibility: VirtualScriptAlias is only available in 1.3.7 and later.

The VirtualScriptAlias directive allows you to determine where Apache will find CGI scripts in a similar manner to [VirtualDocumentRoot](#) does for other documents. It matches requests for URIs starting /cgi-bin/, much like

[ScriptAlias](#) /cgi-bin/ would.

VirtualScriptAliasIP directive

Syntax: VirtualScriptAliasIP *interpolated-directory*

Default: None

Context: server config, virtual host

Status: Extension

Module: mod_vhost_alias

Compatibility: VirtualScriptAliasIP is only available in 1.3.7 and later.

The VirtualScriptAliasIP directive is like the [VirtualScriptAlias](#) directive, except that it uses the IP address of the server end of the connection instead of the server name.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache API notes

These are some notes on the Apache API and the data structures you have to deal with, *etc.* They are not yet nearly complete, but hopefully, they will help you get your bearings. Keep in mind that the API is still subject to change as we gain experience with it. (See the TODO file for what *might* be coming). However, it will be easy to adapt modules to any changes that are made. (We have more modules to adapt than you do).

A few notes on general pedagogical style here. In the interest of conciseness, all structure declarations here are incomplete --- the real ones have more slots that I'm not telling you about. For the most part, these are reserved to one component of the server core or another, and should be altered by modules with caution. However, in some cases, they really are things I just haven't gotten around to yet. Welcome to the bleeding edge.

Finally, here's an outline, to give you some bare idea of what's coming up, and in what order:

- [Basic concepts.](#)
 - [Handlers, Modules, and Requests](#)
 - [A brief tour of a module](#)
- [How handlers work](#)
 - [A brief tour of the `request_rec`](#)
 - [Where `request_rec` structures come from](#)
 - [Handling requests, declining, and returning error codes](#)
 - [Special considerations for response handlers](#)
 - [Special considerations for authentication handlers](#)
 - [Special considerations for logging handlers](#)
- [Resource allocation and resource pools](#)
- [Configuration, commands and the like](#)
 - [Per-directory configuration structures](#)
 - [Command handling](#)
 - [Side notes --- per-server configuration, virtual servers, *etc.*](#)

Basic concepts.

We begin with an overview of the basic concepts behind the API, and how they are manifested in the code.

Handlers, Modules, and Requests

Apache breaks down request handling into a series of steps, more or less the same way the Netscape server API does (although this API has a few more stages than NetSite does, as hooks for stuff I thought might be useful in the future). These are:

- URI -> Filename translation
- Auth ID checking [is the user who they say they are?]
- Auth access checking [is the user authorized *here*?]

- Access checking other than auth
- Determining MIME type of the object requested
- `Fixups' --- there aren't any of these yet, but the phase is intended as a hook for possible extensions like `SetEnv`, which don't really fit well elsewhere.
- Actually sending a response back to the client.
- Logging the request

These phases are handled by looking at each of a succession of *modules*, looking to see if each of them has a handler for the phase, and attempting invoking it if so. The handler can typically do one of three things:

- *Handle* the request, and indicate that it has done so by returning the magic constant `OK`.
- *Decline* to handle the request, by returning the magic integer constant `DECLINED`. In this case, the server behaves in all respects as if the handler simply hadn't been there.
- Signal an error, by returning one of the HTTP error codes. This terminates normal handling of the request, although an `ErrorDocument` may be invoked to try to mop up, and it will be logged in any case.

Most phases are terminated by the first module that handles them; however, for logging, `fixups', and non-access authentication checking, all handlers always run (barring an error). Also, the response phase is unique in that modules may declare multiple handlers for it, via a dispatch table keyed on the MIME type of the requested object. Modules may declare a response-phase handler which can handle *any* request, by giving it the key `/*/*` (*i.e.*, a wildcard MIME type specification). However, wildcard handlers are only invoked if the server has already tried and failed to find a more specific response handler for the MIME type of the requested object (either none existed, or they all declined).

The handlers themselves are functions of one argument (a `request_rec` structure. vide infra), which returns an integer, as above.

A brief tour of a module

At this point, we need to explain the structure of a module. Our candidate will be one of the messier ones, the CGI module --- this handles both CGI scripts and the `ScriptAlias` config file command. It's actually a great deal more complicated than most modules, but if we're going to have only one example, it might as well be the one with its fingers in every place.

Let's begin with handlers. In order to handle the CGI scripts, the module declares a response handler for them. Because of `ScriptAlias`, it also has handlers for the name translation phase (to recognize `ScriptAliased` URIs), the type-checking phase (any `ScriptAliased` request is typed as a CGI script).

The module needs to maintain some per (virtual) server information, namely, the `ScriptAliases` in effect; the module structure therefore contains pointers to a functions which builds these structures, and to another which combines two of them (in case the main server and a virtual server both have `ScriptAliases` declared).

Finally, this module contains code to handle the `ScriptAlias` command itself. This particular module only declares one command, but there could be more, so modules have *command tables* which declare their commands, and describe where they are permitted, and how they are to be invoked.

A final note on the declared types of the arguments of some of these commands: a `pool` is a pointer to a *resource pool* structure; these are used by the server to keep track of the memory which has been allocated, files opened, *etc.*, either to service a particular request, or to handle the process of configuring itself. That way, when the request is over (or, for the configuration pool, when the server is restarting), the memory can be freed, and the files closed, *en masse*, without anyone having to write explicit code to track them all down and dispose of them. Also, a `cmd_parms` structure contains various information about the config file being read, and other status information, which is sometimes of use to the function which processes a config-file command (such as `ScriptAlias`). With no further ado, the module itself:

```
/* Declarations of handlers. */

int translate_scriptalias (request_rec *);
int type_scriptalias (request_rec *);
int cgi_handler (request_rec *);

/* Subsidiary dispatch table for response-phase handlers, by MIME type */

handler_rec cgi_handlers[] = {
```

```

{ "application/x-httpd-cgi", cgi_handler },
{ NULL }
};

/* Declarations of routines to manipulate the module's configuration
 * info. Note that these are returned, and passed in, as void *'s;
 * the server core keeps track of them, but it doesn't, and can't,
 * know their internal structure.
 */

void *make_cgi_server_config (pool *);
void *merge_cgi_server_config (pool *, void *, void *);

/* Declarations of routines to handle config-file commands */

extern char *script_alias(cmd_parms *, void *per_dir_config, char *fake,
                          char *real);

command_rec cgi_cmds[] = {
{ "ScriptAlias", script_alias, NULL, RSRC_CONF, TAKE2,
  "a fakename and a realname"},
{ NULL }
};

module cgi_module = {
  STANDARD_MODULE_STUFF,
  NULL, /* initializer */
  NULL, /* dir config creator */
  NULL, /* dir merger --- default is to override */
  make_cgi_server_config, /* server config */
  merge_cgi_server_config, /* merge server config */
  cgi_cmds, /* command table */
  cgi_handlers, /* handlers */
  translate_scriptalias, /* filename translation */
  NULL, /* check_user_id */
  NULL, /* check_auth */
  NULL, /* check_access */
  type_scriptalias, /* type_checker */
  NULL, /* fixups */
  NULL, /* logger */
  NULL /* header parser */
};

```

How handlers work

The sole argument to handlers is a `request_rec` structure. This structure describes a particular request which has been made to the server, on behalf of a client. In most cases, each connection to the client generates only one `request_rec` structure.

A brief tour of the `request_rec`

The `request_rec` contains pointers to a resource pool which will be cleared when the server is finished handling the request; to structures containing per-server and per-connection information, and most importantly, information on the request itself.

The most important such information is a small set of character strings describing attributes of the object being requested, including its URI, filename, content-type and content-encoding (these being filled in by the translation and type-check handlers which handle the request, respectively).

Other commonly used data items are tables giving the MIME headers on the client's original request, MIME headers to be sent back with the response (which modules can add to at will), and environment variables for any subprocesses which are spawned off in the course of servicing the request. These tables are manipulated using the `ap_table_get` and `ap_table_set` routines.

Note that the Content-type header value *cannot* be set by module content-handlers using the `ap_table_*`() routines. Rather, it is set by pointing the `content_type` field in the `request_rec` structure to an appropriate string. *E.g.*,

```
r->content_type = "text/html";
```

Finally, there are pointers to two data structures which, in turn, point to per-module configuration structures. Specifically, these hold pointers to the data structures which the module has built to describe the way it has been configured to operate in a given directory (via `.htaccess` files or `<Directory>` sections), for private data it has built in the course of servicing the request (so modules' handlers for one phase can pass 'notes' to their handlers for other phases). There is another such configuration vector in the `server_rec` data structure pointed to by the `request_rec`, which contains per (virtual) server configuration data.

Here is an abridged declaration, giving the fields most commonly used:

```
struct request_rec {

    pool *pool;
    conn_rec *connection;
    server_rec *server;

    /* What object is being requested */

    char *uri;
    char *filename;
    char *path_info;
    char *args;          /* QUERY_ARGS, if any */
    struct stat finfo;   /* Set by server core;
                        * st_mode set to zero if no such file */

    char *content_type;
    char *content_encoding;

    /* MIME header environments, in and out. Also, an array containing
     * environment variables to be passed to subprocesses, so people can
     * write modules to add to that environment.
     *
     * The difference between headers_out and err_headers_out is that
     * the latter are printed even on error, and persist across internal
     * redirects (so the headers printed for ErrorDocument handlers will
     * have them).
     */

    table *headers_in;
    table *headers_out;
    table *err_headers_out;
    table *subprocess_env;

    /* Info about the request itself... */

    int header_only;    /* HEAD request, as opposed to GET */
    char *protocol;     /* Protocol, as given to us, or HTTP/0.9 */
    char *method;       /* GET, HEAD, POST, etc. */
    int method_number;  /* M_GET, M_POST, etc. */

    /* Info for logging */

    char *the_request;
    int bytes_sent;

    /* A flag which modules can set, to indicate that the data being
     * returned is volatile, and clients should be told not to cache it.
     */
};
```

```

int no_cache;

/* Various other config info which may change with .htaccess files
 * These are config vectors, with one void* pointer for each module
 * (the thing pointed to being the module's business).
 */

void *per_dir_config; /* Options set in config files, etc. */
void *request_config; /* Notes on *this* request */

};

```

Where request_rec structures come from

Most `request_rec` structures are built by reading an HTTP request from a client, and filling in the fields. However, there are a few exceptions:

- If the request is to an imagemap, a type map (*i.e.*, a `*.var` file), or a CGI script which returned a local ``Location:'`, then the resource which the user requested is going to be ultimately located by some URI other than what the client originally supplied. In this case, the server does an *internal redirect*, constructing a new `request_rec` for the new URI, and processing it almost exactly as if the client had requested the new URI directly.
- If some handler signaled an error, and an `ErrorDocument` is in scope, the same internal redirect machinery comes into play.
- Finally, a handler occasionally needs to investigate ``what would happen if'` some other request were run. For instance, the directory indexing module needs to know what MIME type would be assigned to a request for each directory entry, in order to figure out what icon to use.

Such handlers can construct a *sub-request*, using the functions `ap_sub_req_lookup_file`, `ap_sub_req_lookup_uri`, and `ap_sub_req_method_uri`; these construct a new `request_rec` structure and processes it as you would expect, up to but not including the point of actually sending a response. (These functions skip over the access checks if the sub-request is for a file in the same directory as the original request).

(Server-side includes work by building sub-requests and then actually invoking the response handler for them, via the function `ap_run_sub_req`).

Handling requests, declining, and returning error codes

As discussed above, each handler, when invoked to handle a particular `request_rec`, has to return an `int` to indicate what happened. That can either be

- OK --- the request was handled successfully. This may or may not terminate the phase.
- DECLINED --- no erroneous condition exists, but the module declines to handle the phase; the server tries to find another.
- an HTTP error code, which aborts handling of the request.

Note that if the error code returned is `REDIRECT`, then the module should put a `Location` in the request's `headers_out`, to indicate where the client should be redirected *to*.

Special considerations for response handlers

Handlers for most phases do their work by simply setting a few fields in the `request_rec` structure (or, in the case of access checkers, simply by returning the correct error code). However, response handlers have to actually send a request back to the client.

They should begin by sending an HTTP response header, using the function `ap_send_http_header`. (You don't have to do anything special to skip sending the header for HTTP/0.9 requests; the function figures out on its own that it shouldn't do anything). If the request is marked `header_only`, that's all they should do; they should return after that, without attempting any further output.

Otherwise, they should produce a request body which responds to the client as appropriate. The primitives for this are `ap_rputc` and `ap_rprintf`, for internally generated output, and `ap_send_fd`, to copy the contents of some `FILE *` straight to the client.

At this point, you should more or less understand the following piece of code, which is the handler which handles GET requests which have no more specific handler; it also shows how conditional GETs can be handled, if it's desirable to do so in a particular response handler --- `ap_set_last_modified` checks against the `If-modified-since` value supplied by the client, if any, and returns an appropriate code (which will, if nonzero, be `USE_LOCAL_COPY`). No similar considerations apply for `ap_set_content_length`, but it returns an error code for symmetry.

```
int default_handler (request_rec *r)
{
    int errstatus;
    FILE *f;

    if (r->method_number != M_GET) return DECLINED;
    if (r->finfo.st_mode == 0) return NOT_FOUND;

    if ((errstatus = ap_set_content_length (r, r->finfo.st_size))
        || (errstatus = ap_set_last_modified (r, r->finfo.st_mtime)))
        return errstatus;

    f = fopen (r->filename, "r");

    if (f == NULL) {
        log_reason("file permissions deny server access",
                  r->filename, r);
        return FORBIDDEN;
    }

    register_timeout ("send", r);
    ap_send_http_header (r);

    if (!r->header_only) send_fd (f, r);
    ap_pfclose (r->pool, f);
    return OK;
}
```

Finally, if all of this is too much of a challenge, there are a few ways out of it. First off, as shown above, a response handler which has not yet produced any output can simply return an error code, in which case the server will automatically produce an error response. Secondly, it can punt to some other handler by invoking `ap_internal_redirect`, which is how the internal redirection machinery discussed above is invoked. A response handler which has internally redirected should always return OK.

(Invoking `ap_internal_redirect` from handlers which are *not* response handlers will lead to serious confusion).

Special considerations for authentication handlers

Stuff that should be discussed here in detail:

- Authentication-phase handlers not invoked unless auth is configured for the directory.
- Common auth configuration stored in the core per-dir configuration; it has accessors `ap_auth_type`, `ap_auth_name`, and `ap_requires`.
- Common routines, to handle the protocol end of things, at least for HTTP basic authentication (`ap_get_basic_auth_pw`, which sets the `connection->user` structure field automatically, and `ap_note_basic_auth_failure`, which arranges for the proper `WWW-Authenticate:` header to be sent back).

Special considerations for logging handlers

When a request has internally redirected, there is the question of what to log. Apache handles this by bundling the entire chain of redirects into a list of `request_rec` structures which are threaded through the `r->prev` and `r->next` pointers. The `request_rec` which is passed to the logging handlers in such cases is the one which was originally built for the initial request from the client; note that the `bytes_sent` field will only be correct in the last request in the chain (the one for which a response was actually sent).

Resource allocation and resource pools

One of the problems of writing and designing a server-pool server is that of preventing leakage, that is, allocating resources (memory, open files, *etc.*), without subsequently releasing them. The resource pool machinery is designed to make it easy to prevent this from happening, by allowing resource to be allocated in such a way that they are *automatically* released when the server is done with them.

The way this works is as follows: the memory which is allocated, file opened, *etc.*, to deal with a particular request are tied to a *resource pool* which is allocated for the request. The pool is a data structure which itself tracks the resources in question.

When the request has been processed, the pool is *cleared*. At that point, all the memory associated with it is released for reuse, all files associated with it are closed, and any other clean-up functions which are associated with the pool are run. When this is over, we can be confident that all the resource tied to the pool have been released, and that none of them have leaked.

Server restarts, and allocation of memory and resources for per-server configuration, are handled in a similar way. There is a *configuration pool*, which keeps track of resources which were allocated while reading the server configuration files, and handling the commands therein (for instance, the memory that was allocated for per-server module configuration, log files and other files that were opened, and so forth). When the server restarts, and has to reread the configuration files, the configuration pool is cleared, and so the memory and file descriptors which were taken up by reading them the last time are made available for reuse.

It should be noted that use of the pool machinery isn't generally obligatory, except for situations like logging handlers, where you really need to register cleanups to make sure that the log file gets closed when the server restarts (this is most easily done by using the function `ap_pfdopen`, which also arranges for the underlying file descriptor to be closed before any child processes, such as for CGI scripts, are `execed`), or in case you are using the timeout machinery (which isn't yet even documented here). However, there are two benefits to using it: resources allocated to a pool never leak (even if you allocate a scratch string, and just forget about it); also, for memory allocation, `ap_palloc` is generally faster than `malloc`.

We begin here by describing how memory is allocated to pools, and then discuss how other resources are tracked by the resource pool machinery.

Allocation of memory in pools

Memory is allocated to pools by calling the function `ap_palloc`, which takes two arguments, one being a pointer to a resource pool structure, and the other being the amount of memory to allocate (in chars). Within handlers for handling requests, the most common way of getting a resource pool structure is by looking at the `pool` slot of the relevant `request_rec`; hence the repeated appearance of the following idiom in module code:

```
int my_handler(request_rec *r)
{
    struct my_structure *foo;
    ...

    foo = (foo *)ap_palloc (r->pool, sizeof(my_structure));
}
```

Note that *there is no* `ap_pfree` --- `ap_palloc`d memory is freed only when the associated resource pool is cleared. This means that `ap_palloc` does not have to do as much accounting as `malloc()`; all it does in the typical case is to round up the size, bump a pointer, and do a range check.

(It also raises the possibility that heavy use of `ap_palloc` could cause a server process to grow excessively large. There are two ways to deal with this, which are dealt with below; briefly, you can use `malloc`, and try to be sure that all of the memory gets explicitly freed, or you can allocate a sub-pool of the main pool, allocate your memory in the sub-pool, and clear it out periodically. The latter technique is discussed in the section on sub-pools below, and is used in the directory-indexing code, in order to avoid excessive storage allocation when listing directories with thousands of files).

Allocating initialized memory

There are functions which allocate initialized memory, and are frequently useful. The function `ap_palloc` has the same interface as `ap_palloc`, but clears out the memory it allocates before it returns it. The function `ap_pstrdup` takes a resource pool and a `char *` as arguments, and allocates memory for a copy of the string the pointer points to, returning a pointer to the copy. Finally `ap_pstrcat` is a varargs-style function, which takes a pointer to a resource pool, and at least two `char *` arguments, the last of which must be `NULL`. It allocates enough memory to fit copies of each of the strings, as a unit; for instance:

```
ap_pstrcat (r->pool, "foo", "/", "bar", NULL);
```

returns a pointer to 8 bytes worth of memory, initialized to "foo/bar".

Commonly-used pools in the Apache Web server

A pool is really defined by its lifetime more than anything else. There are some static pools in `http_main` which are passed to various non-`http_main` functions as arguments at opportune times. Here they are:

permanent_pool

- never passed to anything else, this is the ancestor of all pools

pconf

- subpool of `permanent_pool`
- created at the beginning of a config "cycle"; exists until the server is terminated or restarts; passed to all config-time routines, either via `cmd->pool`, or as the "pool *p" argument on those which don't take pools
- passed to the module `init()` functions

ptemp

- sorry I lie, this pool isn't called this currently in 1.3, I renamed it this in my pthreads development. I'm referring to the use of `ptrans` in the parent... contrast this with the later definition of `ptrans` in the child.
- subpool of `permanent_pool`
- created at the beginning of a config "cycle"; exists until the end of config parsing; passed to config-time routines *via* `cmd->temp_pool`. Somewhat of a "bastard child" because it isn't available everywhere. Used for temporary scratch space which may be needed by some config routines but which is deleted at the end of config.

pchild

- subpool of `permanent_pool`
- created when a child is spawned (or a thread is created); lives until that child (thread) is destroyed
- passed to the module `child_init` functions
- destruction happens right after the `child_exit` functions are called... (which may explain why I think `child_exit` is redundant and unneeded)

ptrans

- should be a subpool of `pchild`, but currently is a subpool of `permanent_pool`, see above
- cleared by the child before going into the `accept()` loop to receive a connection
- used as `connection->pool`

r->pool

- for the main request this is a subpool of `connection->pool`; for subrequests it is a subpool of the parent request's pool.
- exists until the end of the request (*i.e.*, `ap_destroy_sub_req`, or in `child_main` after `process_request` has finished)
- note that `r` itself is allocated from `r->pool`; *i.e.*, `r->pool` is first created and then `r` is the first thing `palloc()`d from it

For almost everything folks do, `r->pool` is the pool to use. But you can see how other lifetimes, such as `pchild`, are useful to some modules... such as modules that need to open a database connection once per child, and wish to clean it up when the child dies.

You can also see how some bugs have manifested themselves, such as setting `connection->user` to a value from `r->pool` -- in this case `connection` exists for the lifetime of `ptrans`, which is longer than `r->pool` (especially if `r->pool` is a subrequest!). So the correct thing

to do is to allocate from connection->pool.

And there was another interesting bug in mod_include/mod_cgi. You'll see in those that they do this test to decide if they should use r->pool or r->main->pool. In this case the resource that they are registering for cleanup is a child process. If it were registered in r->pool, then the code would wait() for the child when the subrequest finishes. With mod_include this could be any old #include, and the delay can be up to 3 seconds... and happened quite frequently. Instead the subprocess is registered in r->main->pool which causes it to be cleaned up when the entire request is done -- *i.e.*, after the output has been sent to the client and logging has happened.

Tracking open files, etc.

As indicated above, resource pools are also used to track other sorts of resources besides memory. The most common are open files. The routine which is typically used for this is ap_pfdopen, which takes a resource pool and two strings as arguments; the strings are the same as the typical arguments to fopen, *e.g.*,

```
...
FILE *f = ap_pfdopen (r->pool, r->filename, "r");

if (f == NULL) { ... } else { ... }
```

There is also a ap_popenf routine, which parallels the lower-level open system call. Both of these routines arrange for the file to be closed when the resource pool in question is cleared.

Unlike the case for memory, there *are* functions to close files allocated with ap_pfdopen, and ap_popenf, namely ap_pfclose and ap_pclosef. (This is because, on many systems, the number of files which a single process can have open is quite limited). It is important to use these functions to close files allocated with ap_pfdopen and ap_popenf, since to do otherwise could cause fatal errors on systems such as Linux, which react badly if the same FILE* is closed more than once.

(Using the close functions is not mandatory, since the file will eventually be closed regardless, but you should consider it in cases where your module is opening, or could open, a lot of files).

Other sorts of resources --- cleanup functions

More text goes here. Describe the the cleanup primitives in terms of which the file stuff is implemented; also, spawn_process.

Pool cleanups live until clear_pool() is called: clear_pool(a) recursively calls destroy_pool() on all subpools of a; then calls all the cleanups for a; then releases all the memory for a. destroy_pool(a) calls clear_pool(a) and then releases the pool structure itself. *i.e.*, clear_pool(a) doesn't delete a, it just frees up all the resources and you can start using it again immediately.

Fine control --- creating and dealing with sub-pools, with a note on sub-requests

On rare occasions, too-free use of ap_palloc() and the associated primitives may result in undesirably profligate resource allocation. You can deal with such a case by creating a *sub-pool*, allocating within the sub-pool rather than the main pool, and clearing or destroying the sub-pool, which releases the resources which were associated with it. (This really *is* a rare situation; the only case in which it comes up in the standard module set is in case of listing directories, and then only with *very* large directories. Unnecessary use of the primitives discussed here can hair up your code quite a bit, with very little gain).

The primitive for creating a sub-pool is ap_make_sub_pool, which takes another pool (the parent pool) as an argument. When the main pool is cleared, the sub-pool will be destroyed. The sub-pool may also be cleared or destroyed at any time, by calling the functions ap_clear_pool and ap_destroy_pool, respectively. (The difference is that ap_clear_pool frees resources associated with the pool, while ap_destroy_pool also deallocates the pool itself. In the former case, you can allocate new resources within the pool, and clear it again, and so forth; in the latter case, it is simply gone).

One final note --- sub-requests have their own resource pools, which are sub-pools of the resource pool for the main request. The polite way to reclaim the resources associated with a sub request which you have allocated (using the ap_sub_req_... functions) is ap_destroy_sub_req, which frees the resource pool. Before calling this function, be sure to copy anything that you care about which might be allocated in the sub-request's resource pool into someplace a little less volatile (for instance, the filename in its request_rec structure).

(Again, under most circumstances, you shouldn't feel obliged to call this function; only 2K of memory or so are allocated for a typical sub request, and it will be freed anyway when the main request pool is cleared. It is only when you are allocating many, many sub-requests for a single main request that you should seriously consider the `ap_destroy_...` functions).

Configuration, commands and the like

One of the design goals for this server was to maintain external compatibility with the NCSA 1.3 server --- that is, to read the same configuration files, to process all the directives therein correctly, and in general to be a drop-in replacement for NCSA. On the other hand, another design goal was to move as much of the server's functionality into modules which have as little as possible to do with the monolithic server core. The only way to reconcile these goals is to move the handling of most commands from the central server into the modules.

However, just giving the modules command tables is not enough to divorce them completely from the server core. The server has to remember the commands in order to act on them later. That involves maintaining data which is private to the modules, and which can be either per-server, or per-directory. Most things are per-directory, including in particular access control and authorization information, but also information on how to determine file types from suffixes, which can be modified by `AddType` and `DefaultType` directives, and so forth. In general, the governing philosophy is that anything which *can* be made configurable by directory should be; per-server information is generally used in the standard set of modules for information like `Aliases` and `Redirects` which come into play before the request is tied to a particular place in the underlying file system.

Another requirement for emulating the NCSA server is being able to handle the per-directory configuration files, generally called `.htaccess` files, though even in the NCSA server they can contain directives which have nothing at all to do with access control. Accordingly, after URI -> filename translation, but before performing any other phase, the server walks down the directory hierarchy of the underlying filesystem, following the translated pathname, to read any `.htaccess` files which might be present. The information which is read in then has to be *merged* with the applicable information from the server's own config files (either from the `<Directory>` sections in `access.conf`, or from defaults in `srn.conf`, which actually behaves for most purposes almost exactly like `<Directory />`).

Finally, after having served a request which involved reading `.htaccess` files, we need to discard the storage allocated for handling them. That is solved the same way it is solved wherever else similar problems come up, by tying those structures to the per-transaction resource pool.

Per-directory configuration structures

Let's look out how all of this plays out in `mod_mime.c`, which defines the file typing handler which emulates the NCSA server's behavior of determining file types from suffixes. What we'll be looking at, here, is the code which implements the `AddType` and `AddEncoding` commands. These commands can appear in `.htaccess` files, so they must be handled in the module's private per-directory data, which in fact, consists of two separate tables for MIME types and encoding information, and is declared as follows:

```
typedef struct {
    table *forced_types;      /* Additional AddTyped stuff */
    table *encoding_types;   /* Added with AddEncoding... */
} mime_dir_config;
```

When the server is reading a configuration file, or `<Directory>` section, which includes one of the MIME module's commands, it needs to create a `mime_dir_config` structure, so those commands have something to act on. It does this by invoking the function it finds in the module's 'create per-dir config slot', with two arguments: the name of the directory to which this configuration information applies (or `NULL` for `srn.conf`), and a pointer to a resource pool in which the allocation should happen.

(If we are reading a `.htaccess` file, that resource pool is the per-request resource pool for the request; otherwise it is a resource pool which is used for configuration data, and cleared on restarts. Either way, it is important for the structure being created to vanish when the pool is cleared, by registering a cleanup on the pool if necessary).

For the MIME module, the per-dir config creation function just `ap_pallocs` the structure above, and a creates a couple of tables to fill it. That looks like this:

```
void *create_mime_dir_config (pool *p, char *dummy)
{
```

```

mime_dir_config *new =
    (mime_dir_config *) ap_palloc (p, sizeof(mime_dir_config));

new->forced_types = ap_make_table (p, 4);
new->encoding_types = ap_make_table (p, 4);

return new;
}

```

Now, suppose we've just read in a `.htaccess` file. We already have the per-directory configuration structure for the next directory up in the hierarchy. If the `.htaccess` file we just read in didn't have any `AddType` or `AddEncoding` commands, its per-directory config structure for the MIME module is still valid, and we can just use it. Otherwise, we need to merge the two structures somehow.

To do that, the server invokes the module's per-directory config merge function, if one is present. That function takes three arguments: the two structures being merged, and a resource pool in which to allocate the result. For the MIME module, all that needs to be done is overlay the tables from the new per-directory config structure with those from the parent:

```

void *merge_mime_dir_configs (pool *p, void *parent_dirv, void *subdirv)
{
    mime_dir_config *parent_dir = (mime_dir_config *)parent_dirv;
    mime_dir_config *subdir = (mime_dir_config *)subdirv;
    mime_dir_config *new =
        (mime_dir_config *)ap_palloc (p, sizeof(mime_dir_config));

    new->forced_types = ap_overlay_tables (p, subdir->forced_types,
                                          parent_dir->forced_types);
    new->encoding_types = ap_overlay_tables (p, subdir->encoding_types,
                                           parent_dir->encoding_types);

    return new;
}

```

As a note --- if there is no per-directory merge function present, the server will just use the subdirectory's configuration info, and ignore the parent's. For some modules, that works just fine (*e.g.*, for the `includes` module, whose per-directory configuration information consists solely of the state of the `XBITHACK`), and for those modules, you can just not declare one, and leave the corresponding structure slot in the module itself `NULL`.

Command handling

Now that we have these structures, we need to be able to figure out how to fill them. That involves processing the actual `AddType` and `AddEncoding` commands. To find commands, the server looks in the module's `command` table. That table contains information on how many arguments the commands take, and in what formats, where it is permitted, and so forth. That information is sufficient to allow the server to invoke most command-handling functions with pre-parsed arguments. Without further ado, let's look at the `AddType` command handler, which looks like this (the `AddEncoding` command looks basically the same, and won't be shown here):

```

char *add_type(cmd_parms *cmd, mime_dir_config *m, char *ct, char *ext)
{
    if (*ext == '.') ++ext;
    ap_table_set (m->forced_types, ext, ct);
    return NULL;
}

```

This command handler is unusually simple. As you can see, it takes four arguments, two of which are pre-parsed arguments, the third being the per-directory configuration structure for the module in question, and the fourth being a pointer to a `cmd_parms` structure. That structure contains a bunch of arguments which are frequently of use to some, but not all, commands, including a resource pool (from which memory can be allocated, and to which cleanups should be tied), and the (virtual) server being configured, from which the module's per-server configuration data can be obtained if required.

Another way in which this particular command handler is unusually simple is that there are no error conditions which it can encounter. If there were, it could return an error message instead of `NULL`; this causes an error to be printed out on the server's

`stderr`, followed by a quick exit, if it is in the main config files; for a `.htaccess` file, the syntax error is logged in the server error log (along with an indication of where it came from), and the request is bounced with a server error response (HTTP error status, code 500).

The MIME module's command table has entries for these commands, which look like this:

```
command_rec mime_cmds[] = {
{ "AddType", add_type, NULL, OR_FILEINFO, TAKE2,
  "a mime type followed by a file extension" },
{ "AddEncoding", add_encoding, NULL, OR_FILEINFO, TAKE2,
  "an encoding (e.g., gzip), followed by a file extension" },
{ NULL }
};
```

The entries in these tables are:

- The name of the command
- The function which handles it
- a `(void *)` pointer, which is passed in the `cmd_parms` structure to the command handler --- this is useful in case many similar commands are handled by the same function.
- A bit mask indicating where the command may appear. There are mask bits corresponding to each `AllowOverride` option, and an additional mask bit, `RSRC_CONF`, indicating that the command may appear in the server's own config files, but *not* in any `.htaccess` file.
- A flag indicating how many arguments the command handler wants pre-parsed, and how they should be passed in. `TAKE2` indicates two pre-parsed arguments. Other options are `TAKE1`, which indicates one pre-parsed argument, `FLAG`, which indicates that the argument should be `On` or `Off`, and is passed in as a boolean flag, `RAW_ARGS`, which causes the server to give the command the raw, unparsed arguments (everything but the command name itself). There is also `ITERATE`, which means that the handler looks the same as `TAKE1`, but that if multiple arguments are present, it should be called multiple times, and finally `ITERATE2`, which indicates that the command handler looks like a `TAKE2`, but if more arguments are present, then it should be called multiple times, holding the first argument constant.
- Finally, we have a string which describes the arguments that should be present. If the arguments in the actual config file are not as required, this string will be used to help give a more specific error message. (You can safely leave this `NULL`).

Finally, having set this all up, we have to use it. This is ultimately done in the module's handlers, specifically for its file-typing handler, which looks more or less like this; note that the per-directory configuration structure is extracted from the `request_rec`'s per-directory configuration vector by using the `ap_get_module_config` function.

```
int find_ct(request_rec *r)
{
    int i;
    char *fn = ap_pstrdup (r->pool, r->filename);
    mime_dir_config *conf = (mime_dir_config *)
        ap_get_module_config(r->per_dir_config, &mime_module);
    char *type;

    if (S_ISDIR(r->finfo.st_mode)) {
        r->content_type = DIR_MAGIC_TYPE;
        return OK;
    }

    if((i=ap_rind(fn, '.')) < 0) return DECLINED;
    ++i;

    if ((type = ap_table_get (conf->encoding_types, &fn[i])))
    {
        r->content_encoding = type;

        /* go back to previous extension to try to use it as a type */

        fn[i-1] = '\0';
        if((i=ap_rind(fn, '.')) < 0) return OK;
```

```

    ++i;
}

if ((type = ap_table_get (conf->forced_types, &fn[i])))
{
    r->content_type = type;
}

return OK;
}

```

Side notes --- per-server configuration, virtual servers, etc.

The basic ideas behind per-server module configuration are basically the same as those for per-directory configuration; there is a creation function and a merge function, the latter being invoked where a virtual server has partially overridden the base server configuration, and a combined structure must be computed. (As with per-directory configuration, the default if no merge function is specified, and a module is configured in some virtual server, is that the base configuration is simply ignored).

The only substantial difference is that when a command needs to configure the per-server private module data, it needs to go to the `cmd_parms` data to get at it. Here's an example, from the alias module, which also indicates how a syntax error can be returned (note that the per-directory configuration argument to the command handler is declared as a dummy, since the module doesn't actually have per-directory config data):

```

char *add_redirect(cmd_parms *cmd, void *dummy, char *f, char *url)
{
    server_rec *s = cmd->server;
    alias_server_conf *conf = (alias_server_conf *)
        ap_get_module_config(s->module_config, &alias_module);
    alias_entry *new = ap_push_array (conf->redirects);

    if (!ap_is_url (url)) return "Redirect to non-URL";

    new->fake = f; new->real = url;
    return NULL;
}

```

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Apache Miscellaneous Documentation

Below is a list of additional documentation pages that apply to the Apache web server development project.

[API](#)

Description of Apache's Application Programming Interface.

[FAQ](#)

Frequently-Asked Questions concerning the Apache project and server.

[Reading Client Input in Apache 1.2](#)

Describes differences between Apache 1.1 and 1.2 in how modules read information from the client.

[Compatibility with NCSA](#)

Notes about Apache's compatibility with the NCSA server.

[How to use XSSI and Negotiation for custom ErrorDocuments](#)

Describes a solution which uses XSSI and negotiation to custom-tailor the Apache ErrorDocuments to taste, adding the advantage of returning internationalized versions of the error messages depending on the client's language preferences.

[File Descriptor use in Apache](#)

Describes how Apache uses file descriptors and talks about various limits imposed on the number of descriptors available by various operating systems.

[FIN_WAIT_2](#)

A description of the causes of Apache processes going into the FIN_WAIT_2 state, and what you can do about it.

["How-To"](#)

Instructions about how to accomplish some commonly-desired server functionality changes.

[HTTP Features list](#)

A tab-separate table of HTTP features implemented and tested in Apache.

[Known Client Problems](#)

A list of problems in HTTP clients which can be mitigated by Apache.

[No PGP](#)

Why we took PEM and PGP support out of the base Apache distribution.

[Performance Notes \(BSD 4.4\)](#)

Some notes about ways to improve/optimize Apache performance on BSD 4.4 systems.

[Performance Notes \(Digital UNIX\)](#)

Extracts of USENET postings describing how to optimize Apache performance on Digital UNIX systems.

[Performance Notes \(HPUX\)](#)

Email from an HP engineer on how to optimize HP-UX 10.20.

[Performance Notes \(General\)](#)

Some generic notes about how to improve the performance of your machine/OS.

[Performance Notes -- Apache Tuning](#)

Notes about how to (run-time and compile-time) configure Apache for highest performance. Notes explaining why Apache does some things, and why it doesn't do other things (which make it slower/faster).

[Security Tips](#)

Some "do"s - and "don't"s - for keeping your Apache web site secure.

[Virtual Hosts \(IP-based\)](#)

Excerpts and notes about configuring and using Apache IP-based virtual hosts.

[Windows Bug with Web Keepalive](#)

A brief description of a known problem with Microsoft Windows and web sites accessed using keepalive connections.

Apache HTTP Server Version 1.3





Apache HTTP Server

PATH_INFO Changes in the CGI Environment

Overview

As implemented in Apache 1.1.1 and earlier versions, the method Apache used to create `PATH_INFO` in the CGI environment was counterintuitive, and could result in crashes in certain cases. In Apache 1.2 and beyond, this behavior has changed. Although this results in some compatibility problems with certain legacy CGI applications, the Apache 1.2 behavior is still compatible with the CGI/1.1 specification, and CGI scripts can be easily modified ([see below](#)).

The Problem

Apache 1.1.1 and earlier implemented the `PATH_INFO` and `SCRIPT_NAME` environment variables by looking at the filename, not the URL. While this resulted in the correct values in many cases, when the filesystem path was overloaded to contain path information, it could result in errant behavior. For example, if the following appeared in a config file:

```
Alias /cgi-ralph /usr/local/httpd/cgi-bin/user.cgi/ralph
```

In this case, `user.cgi` is the CGI script, the `/ralph` is information to be passed onto the CGI. If this configuration was in place, and a request came for `/cgi-ralph/script/`, the code would set `PATH_INFO` to `/ralph/script`, and `SCRIPT_NAME` to `/cgi-`. Obviously, the latter is incorrect. In certain cases, this could even cause the server to crash.

The Solution

Apache 1.2 and later now determine `SCRIPT_NAME` and `PATH_INFO` by looking directly at the URL, and determining how much of the URL is client-modifiable, and setting `PATH_INFO` to it. To use the above example, `PATH_INFO` would be set to `/script`, and `SCRIPT_NAME` to `/cgi-ralph`. This makes sense and results in no server behavior problems. It also permits the script to be guaranteed that `http://$SERVER_NAME:$SERVER_PORT$SCRIPT_NAME$PATH_INFO` will always be an accessible URL that points to the current script, something which was not necessarily true with previous versions of Apache.

However, the `/ralph` information from the `Alias` directive is lost. This is unfortunate, but we feel that using the filesystem to pass along this sort of information is not a recommended method, and a script making use of it "deserves" not to work. Apache 1.2b3 and later, however, do provide [a workaround](#).

Compatibility with Previous Servers

It may be necessary for a script that was designed for earlier versions of Apache or other servers to need the information that the old `PATH_INFO` variable provided. For this purpose, Apache 1.2 (1.2b3 and later) sets an additional variable, `FILEPATH_INFO`. This environment variable contains the value that `PATH_INFO` would have had with Apache 1.1.1.

A script that wishes to work with both Apache 1.2 and earlier versions can simply test for the existence of `FILEPATH_INFO`, and use it if available. Otherwise, it can use `PATH_INFO`. For example, in Perl, one might use:

```
$path_info = $ENV{'FILEPATH_INFO'} || $ENV{'PATH_INFO'};
```

By doing this, a script can work with all servers supporting the CGI/1.1 specification, including all versions of Apache.

Apache HTTP Server





Apache HTTP Server

Apache Keep-Alive Support

What is Keep-Alive?

The Keep-Alive extension to HTTP, as defined by the HTTP/1.1 draft, allows persistent connections. These long-lived HTTP sessions allow multiple requests to be sent over the same TCP connection, and in some cases have been shown to result in an almost 50% speedup in latency times for HTML documents with lots of images.

Enabling Keep-Alive Support

Apache 1.1 comes with Keep-Alive support on by default, however there are some directives you can use to modify Apache's behavior:

Note: Apache 1.2 uses a different syntax for the [KeepAlive](#) directive.

KeepAlive

Syntax: `KeepAlive max-requests`

Default: `KeepAlive 5`

Context: server config

Status: Core

This directive enables Keep-Alive support. Set *max-requests* to the maximum number of requests you want Apache to entertain per connection. A limit is imposed to prevent a client from hogging your server resources. Set this to 0 to disable support.

KeepAliveTimeout

Syntax: `KeepAliveTimeout seconds`

Default: `KeepAliveTimeout 15`

Context: server config

Status: Core

The number of seconds Apache will wait for a subsequent request before closing the connection. Once a request has been received, the timeout value specified by the [Timeout](#) directive applies.

When Keep-Alive Is Used

In order for Keep-Alive support to be used, first the browser must support it. Many current browsers, including Netscape Navigator 2.0, and Spyglass Mosaic-based browsers (including Microsoft Internet Explorer) do. Note, however, that some Windows 95-based browsers misbehave with Keep-Alive-supporting servers; they may occasionally hang on a connect. This has been observed with several Windows browsers, and occurs when connecting to any Keep-Alive server, not just Apache. Netscape 3.0b5 and later versions are known to work around this problem.

However, Keep-Alive support only is active with files where the length is known beforehand. This means that most CGI scripts, server-side included files and directory listings will not use the Keep-Alive protocol. While this should be completely transparent to the end user, it is something the web-master may want to keep in mind.

Apache HTTP Server





Apache HTTP Server

Multiple Log Files

It is now possible to specify multiple log files, each with a fully customizable format. This is compatible with existing configurations. Multiple log files are implemented as part of the [mod_log_config](#) module which as of Apache 1.2 is the default log module.

Using Multiple Log Files

Multiple log files be created with either the `TransferLog` or `CustomLog` directive. These directives can be repeated to create more than one log file (in previous releases, only one logfile could be given per server configuration). The `TransferLog` directive creates a log file in the standard "common log format", although this can be customized with `LogFormat`. The syntax of these two directives is the same as for the config log module in previous Apache releases.

The real power of multiple log files come from the ability to create log files in different formats. For example, as well as a CLF transfer log, the server could log the user agent of each client, or the referrer information, or any other aspect of the request, such as the language preferences of the user.

The new `CustomLog` directive takes both a filename to log to, and a log file format.

Syntax: `CustomLog filename "format"`

Context: server config, virtual host

Status: base

Module: `mod_log_config`

The first argument is the filename to log to. This is used exactly like the argument to `TransferLog`, that is, it is either a file as a full path or relative to the current server root, or `|programname`. Be aware that anyone who can write to the directory where a log file is written can gain access to the uid that starts the server. See the [security tips](#) document for details.

The format argument specifies a format for each line of the log file. The options available for the format are exactly the same as for the argument of the `LogFormat` directive. If the format includes any spaces (which it will do in almost all cases) it should be enclosed in double quotes.

Use with Virtual Hosts

If a `<VirtualHost>` section does not contain any `TransferLog` or `CustomLog` directives, the logs defined for the main server will be used. If it does contain one or more of these directives, requests serviced by this virtual host will only be logged in the log files defined within its definition, not in any of the main server's log files. See the examples below.

Examples

To create a normal (CLF) format log file in `logs/access_log`, and a log of user agents:

```
TransferLog logs/access_log
CustomLog logs/agents "%{user-agent}i"
```

To define a CLF transfer log and a referrer log which log all accesses to both the main server and a virtual host:

```
TransferLog logs/access_log  
CustomLog logs/referer "%{referer}i"
```

```
<VirtualHost>  
  DocumentRoot /whatever  
  ServerName my.virtual.host  
</VirtualHost>
```

Since no `TransferLog` or `CustomLog` directives appear inside the `<VirtualHost>` section, any requests for this virtual host will be logged in the main server's log files. If however the directive

```
TransferLog logs/vhost_access_log
```

was added inside the virtual host definition, then accesses to this virtual host will be logged in `vhost_access_log` file (in common log format), and *not* in `logs/access_log` or `logs/referer`.

Apache HTTP Server





Apache HTTP Server

Server Pool Management

We found that many people were using values for "MaxServers" either too high or too low, and were hanging themselves on it. The model we adopted is still based on long-lived minimal-forking processes, but instead of specifying one number of persistent processes, the web-master specifies a maximum and minimum number of processes to be "spare" - every couple of seconds the parent checks the actual number of spare servers and adjusts accordingly. This should keep the number of servers concurrently running relatively low while still ensuring minimal forking.

We renamed the current StartServers to MinSpareServers, created separate StartServers parameter which means what it says, and renamed MaxServers to MaxSpareServers (though the old name still works, for NCSA 1.4 back-compatibility). The old names were generally regarded as too confusing.

The defaults for each variable are:

MinSpareServers	5
MaxSpareServers	10
StartServers	5

There is an absolute maximum number of simultaneous children defined by a compile-time limit which defaults to 256 and a "MaxClients" directive which specifies the number of simultaneous children that will be allowed. MaxClients can be adjusted up to the compile-time limit (HARD_SERVER_LIMIT, defined in httpd.h). If you need more than 256 simultaneous children, you need to modify both HARD_SERVER_LIMIT and MaxClients.

In versions before 1.2, HARD_SERVER_LIMIT defaulted to 150.

We do not recommend changing either of these values unless:

1. You know you have the server resources to handle more
 2. You use the machine for other purposes and must limit the amount of memory Apache uses
-

Apache HTTP Server





Apache HTTP Server

Source Re-organisation

As of 1.3, the Apache source directories have been re-organised. This re-organisation is designed to simplify the directory structure, make it easier to add additional modules, and to give module authors a way of specifying compile time options or distribute binary modules.

Summary of Changes

The source changes are:

- The non-module source files have moved from `src` into `src/main`
- The module source files previously in `src` have moved to `src/modules/standard`
- The `support` directory is now in `src/support`
- The existing symbol names used for global Apache function and variable identifiers have been renamed in the source. This way namespace conflicts are avoided when linking Apache with third-party libraries. See the file `src/include/compat.h` both for the list of renamed symbol names and for a way to get source backward compatibility in existing third-party module sources.

In addition, the following enhancements have been made:

- OS abstractions can be added in the `src/os` directory. Currently this contains information for unix, OS/2 and Windows 32 platforms.
- Configuration syntax has been simplified for adding new modules. Users no longer need to enter the module's structure name. In addition, new modules can be located anywhere on the file system, or typically in new or existing directories under `src/modules`.
- Module authors can give simpler instructions for adding their modules to Apache compilation. They can also now provide compile time information required by `Configure`, such as additional libraries required.
- Module authors can distribute pre-compiled (.a or .o) versions of their modules if required, along with a "module definition file" which contains the information required by `Configure`.
- All the sub-directories (`main`, `modules/*` and `os/*`) are built as libraries.
- The new Apache Autoconf-style Interface (APACI) script named `configure` replaced the old top-level `Makefile` and `src/helpers/InstallApache` stuff.

Adding Modules

Modules are added to Apache by adding a reference to them in `src/Configuration` then running `Configure` and `make`. In earlier version of Apache before 1.3, the line added to `Configuration` looked like this:

```
Module      status_module      mod_status.o
```

From 1.3 onwards, the `AddModule` line should be used instead, and typically looks like this:

```
AddModule  modules/standard/mod_status.o
```

The argument to `AddModule` is the path, relative to `src`, to the module file's source or object file.

Normally when adding a module you should follow the instructions of the module author. However if the module comes as a single source file, say `mod_foo.c`, then the recommended way to add the module to Apache is as follows:

- Put `mod_foo.c` into the directory `src/modules/extra`
- Go to the `src` directory and add the following line to `Configuration`
`AddModule modules/extra/mod_foo.o`
- Run `./Configure`
- Run `make`

New Facilities for Module Authors

In previous releases of Apache, new modules were added to the `src` directory, and if the module required any additional compilation options (such as libraries) they would have to be added to `Configuration`. Also the user would have to be told the module's structure name to add on the `Module` line of `Configuration`.

From Apache 1.3 onwards, module authors can make use of these new features:

- Simplified `Configuration` command `AddModule` which only requires a path to the module source or object file
- If the module requires compile time options (such as extra libraries) these can be specified in the module file source or an external "module definition file".
- If a module is distributed as binary (`.o` or `.a`) then an external "module definition file" can also be distributed which gives the information `Configure` needs to add the module, such as extra libraries and the module's structure name.
- Modules can be installed anywhere on the file system, although a directory under `src/modules` is recommended.
- If the module is in its own directory, Apache can automatically create a `Makefile` to build the module given a file containing the module's dependencies.
- For building a third-party module **outside** the Apache source tree the new `apxs` support tool can be used to compile the module into a [dynamic shared object \(DSO\)](#), install it into the existing Apache installation and optionally activating it in the Apache `httpd.conf` file. The only requirement is that Apache has DSO-support for the used platform and the module `mod_so` was built into the server binary `httpd`.

The rest of this document shows how to package modules for Apache 1.3 and later and what to tell end-users of the module.

Building a simple source distribution

Consider a simple add-on module, distributed as a single file. For example, say it is called `mod_demo.c`. The archive for this module should consist of two files, in a suitable directory name. For example:

- `mod_demo/mod_demo.c`
- `mod_demo/Makefile.tmpl`

(Of course end-user instructions, `README`'s, etc can also be supplied in the archive). The end user should be told to extract this archive in the `src/modules` directory of their Apache source tree. This will create a new directory `src/modules/mod_demo`. Then they need to add the following line to the `Configuration` file:

```
AddModule modules/mod_demo/mod_demo.o
```

then run `Configure` and `make` as normal.

The `mod_demo/Makefile.tmpl` should contain the dependencies of the module source. For example, a simple module which just interfaces to some standard Apache module API functions might look this this:

```
mod_demo.o: mod_demo.c $(INCDIR)/httpd.h $(INCDIR)/http_protocol.h
```

When the user runs `Configure` Apache will create a full makefile to build this module. If this module also requires some additional built-time options to be given, such as libraries, see the next section.

If the module also comes with header files, these can be added to the archive. If the module consists of multiple source files it can be built into a library file using a supplied makefile. In this case, distribute the makefile as `mod_demo/Makefile` and **do not** include a `mod_demo/Makefile.tmpl`. If `Configure` sees a `Makefile.tmpl` it assumes it is safe to overwrite any existing `Makefile`.

See the Apache `src/modules/standard` for an example of a module directory where the makefile is created automatically from a `Makefile.tmpl` file (note that this directory also shows how to distribute multiple modules in a single directory). See `src/modules/proxy` and `src/modules/example` for examples of modules built using custom makefiles (to build a library and an object file, respectively).

Adding Compile time Information

Apache source files (or module definition files, see below) can contain information used by `Configure` to add compile-time options such as additional libraries. For example, if `mod_demo` in the example above also requires that Apache be linked against a DBM library, then the following text could be inserted into the `mod_demo.c` source:

```
/*
 * Module definition information - the part between the -START and -END
 * lines below is used by Configure. This could be stored in a separate
 * instead.
 *
 * MODULE-DEFINITION-START
 * Name: demo_module
 * ConfigStart
 *     LIBS="$LIBS $DBM_LIB"
 *     if [ "X$DBM_LIB" != "X" ]; then
 *         echo " + using $DBM_LIB for mod_demo"
 *     fi
 * ConfigEnd
 * MODULE-DEFINITION-END
 */
```

Note that this is contained inside a C language comment to hide it from the compiler. Anything between the lines which contains `MODULE-DEFINITION-START` and `MODULE-DEFINITION-END` is used by `Configure`. The `Name:` line gives the module's structure name. This is not really necessary in this case since if not present `Configure` will guess at a name based on the filename (*e.g.*, given `"mod_demo"` it will remove the leading `"mod_"` and append `"_module"` to get a structure name. This works with all modules distributed with Apache).

The lines between `ConfigStart` and `ConfigEnd` as executed by `Configure` and can be used to add compile-time options and libraries. In this case it adds the DBM library (from `$DBM_LIB`) to the standard compilation libraries (`$LIB`) and displays a message.

See the default distribution's `mod_auth_dbm.c` for an example of an embedded module definition.

Module Definition Information for Binary Distributions

If the module is to be distributed as binary (object or library) rather than source, it is not possible to add the module definition information to the source file. In this case it can be placed in a separate file which has the same base name as the object or library file, but with a `.module` extension. So, for example, if the distributed module object file is `mod_demo.o`, the module definition file should be called `mod_demo.module`. It contains the same information as above, but does not need to be inside a C comment or delimited with `MODULE-DEFINITION-START` *etc.* For example:

```
Name: demo_module
ConfigStart
LIBS="$LIBS $DBM_LIB"
if [ "X$DBM_LIB" != "X" ]; then
    echo " + using $DBM_LIB for mod_demo"
fi
ConfigEnd
```

See the default distribution's `mod_auth_db.module` for an example of a separate module definition file.

Apache HTTP Server





Apache HTTP Server Version 1.3

Compatibility Notes with NCSA's Server

While Apache is for the most part a drop-in replacement for NCSA's httpd, there are a couple gotcha's to watch out for. These are mostly due to the fact that the parser for config and access control files was rewritten from scratch, so certain liberties the earlier servers took may not be available here. These are all easily fixable. If you know of other problems that belong here, [let us know](#).

Please also check the [known client problems](#) page.

1. As of Apache 1.3.1, methods named in a `<Limit>` section *must* be listed in upper-case. Lower- or mixed-case method names will result in a configuration error.
2. The basic `mod_auth AuthGroupFile`-specified group file format allows commas between user names - Apache does not.
3. If you follow the NCSA guidelines for setting up access restrictions based on client domain, you may well have added entries for `AuthType`, `AuthName`, `AuthUserFile` or `AuthGroupFile`. **None** of these are needed (or appropriate) for restricting access based on client domain. When Apache sees `AuthType` it (reasonably) assumes you are using some authorization type based on username and password. Please remove `AuthType`, it's unnecessary even for NCSA.
4. `OldScriptAlias` is no longer supported.
5. `exec cgi=" "` produces reasonable **malformed header** responses when used to invoke non-CGI scripts. The NCSA code ignores the missing header (bad idea).
Solution: write CGI to the CGI spec and use `include virtual`, or use `exec cmd=" "` instead.
6. Icons for FancyIndexing broken - well, no, they're not broken, we've just upgraded the icons from flat .xbm files to pretty and much smaller .gif files, courtesy of [Kevin Hughes](#) at [EIT](#). If you are using the same srm.conf from an old distribution, make sure you add the new [AddIcon](#), [AddIconByType](#), and [DefaultIcon](#) directives.
7. Apache versions before 1.2b1 will ignore the last line of configuration files if the last line does not have a trailing newline. This affects configuration files (httpd.conf, access.conf and srm.conf), and htpasswd and htgroup files.
8. Apache does not permit commas delimiting the methods in `<Limit>`.
9. Apache's `<VirtualHost>` treats all addresses as "optional" (*i.e.*, the server should continue booting if it can't resolve the address). Whereas in NCSA the default is to fail booting unless an added `optional` keyword is included.
10. Apache does not implement `OnDeny`; use [ErrorDocument](#) instead.
11. Apache (as of 1.3) always performs the equivalent of `HostnameLookups minimal`. `minimal` is not an option to [HostnameLookups](#).
12. To embed spaces in directive arguments NCSA used a backslash before the space. Apache treats backslashes as normal characters. To embed spaces surround the argument with double-quotes instead.
13. Apache does not implement the NCSA `referer` directive. See [PR#968](#) for a few brief suggestions on alternative ways to implement the same thing under Apache.
14. Apache does not allow `ServerRoot` settings inside a `VirtualHost` container. There is only one global `ServerRoot` in Apache; any desired changes in paths for virtual hosts need to be made with the explicit directives, *e.g.*, `DocumentRoot`, `TransferLog`, *etc.*
15. The `AddType` directive cannot be used to set the type of particular files. Instead, you can scope you directives using `<Files>` blocks.

More to come when we notice them....

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Using XSSI and ErrorDocument to configure customized international server error responses

Index

- [Introduction](#)
 - [Creating an ErrorDocument directory](#)
 - [Naming the individual error document files](#)
 - [The common header and footer files](#)
 - [Creating ErrorDocuments in different languages](#)
 - [The fallback language](#)
 - [Customizing Proxy Error Messages](#)
 - [HTML listing of the discussed example](#)
-

Introduction

This document describes an easy way to provide your apache WWW server with a set of customized error messages which take advantage of [Content Negotiation](#) and [eXtended Server Side Includes \(XSSI\)](#) to return error messages generated by the server in the client's native language.

By using XSSI, all [customized messages](#) can share a homogenous and consistent style and layout, and maintenance work (changing images, changing links) is kept to a minimum because all layout information can be kept in a single file.

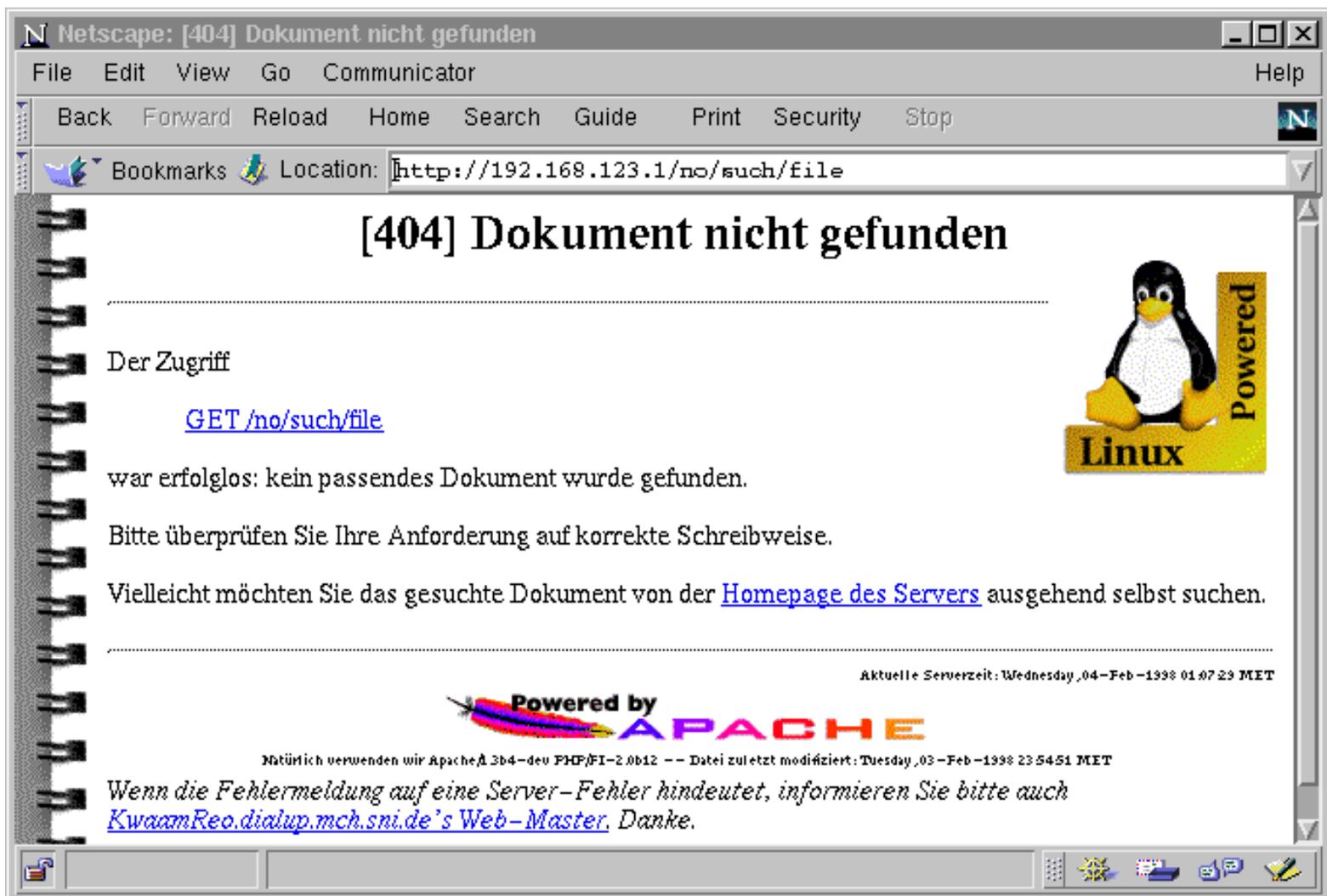
Error documents can be shared across different servers, or even hosts, because all varying information is inserted at the time the error document is returned on behalf of a failed request.

Content Negotiation then selects the appropriate language version of a particular error message text, honoring the language preferences passed in the client's request. (Users usually select their favorite languages in the preferences options menu of today's browsers). When an error document in the client's primary language version is unavailable, the secondary languages are tried or a default (fallback) version is used.

You have full flexibility in designing your error documents to your personal taste (or your company's conventions). For demonstration purposes, we present a simple generic error document scheme. For this hypothetical server, we assume that all error messages...

- possibly are served by different virtual hosts (different host name, different IP address, or different port) on the server machine,
- show a predefined company logo in the right top of the message (selectable by virtual host),
- print the error title first, followed by an explanatory text and (depending on the error context) help on how to resolve the error,
- have some kind of standardized background image,
- display an apache logo and a feedback email address at the bottom of the error message.

An example of a "document not found" message for a german client might look like this:



All links in the document as well as links to the server's administrator mail address, and even the name and port of the serving virtual host are inserted in the error document at "run-time", *i.e.*, when the error actually occurs.

Creating an ErrorDocument directory

For this concept to work as easily as possible, we must take advantage of as much server support as we can get:

1. By defining the [MultiViews option](#), we enable the language selection of the most appropriate language alternative (content negotiation).
2. By setting the [LanguagePriority](#) directive we define a set of default fallback languages in the situation where the client's browser did not express any preference at all.
3. By enabling [Server Side Includes](#) (and disallowing execution of cgi scripts for security reasons), we allow the server to include building blocks of the error message, and to substitute the value of certain environment variables into the generated document (dynamic HTML) or even to conditionally include or omit parts of the text.
4. The [AddHandler](#) and [AddType](#) directives are useful for automatically XSSI-expanding all files with a .shtml suffix to *text/html*.
5. By using the [Alias](#) directive, we keep the error document directory outside of the document tree because it can be regarded more as a server part than part of the document tree.
6. The [<Directory>](#)-Block restricts these "special" settings to the error document directory and avoids an impact on any of the settings for the regular document tree.
7. For each of the error codes to be handled (see RFC2068 for an exact description of each error code, or look at `src/main/http_protocol.c` if you wish to see apache's standard messages), an [ErrorDocument](#) in the aliased `/errordocs` directory is defined. Note that we only define the basename of the document here because the MultiViews option will select the best candidate based on the language suffixes and the client's preferences. Any error situation with an error code *not* handled by a custom document will be dealt with by the server in the standard way (*i.e.*, a plain error message in english).
8. Finally, the [AllowOverride](#) directive tells apache that it is not necessary to look for a .htaccess file in the `/errordocs` directory: a minor speed optimization.

The resulting `httpd.conf` configuration would then look similar to this: (Note that you can define your own error messages using this method for only part of the document tree, e.g., a `/~user/` subtree. In this case, the configuration could as well be put into the .htaccess file at the root of the subtree, and the `<Directory>` and `</Directory>` directives -but not the contained directives- must be omitted.)

```

LanguagePriority en fr de
Alias /errordocs /usr/local/apache/errordocs
<Directory /usr/local/apache/errordocs>
  AllowOverride none
  Options MultiViews IncludesNoExec FollowSymLinks
  AddType text/html .shtml
  AddHandler server-parsed .shtml
</Directory>
#   "400 Bad Request",
ErrorDocument 400 /errordocs/400
#   "401 Authorization Required",
ErrorDocument 401 /errordocs/401
#   "403 Forbidden",
ErrorDocument 403 /errordocs/403
#   "404 Not Found",
ErrorDocument 404 /errordocs/404
#   "500 Internal Server Error",
ErrorDocument 500 /errordocs/500

```

The directory for the error messages (here: /usr/local/apache/errordocs/) must then be created with the appropriate permissions (readable and executable by the server uid or gid, only writable for the administrator).

Naming the individual error document files

By defining the MultiViews option, the server was told to automatically scan the directory for matching variants (looking at language and content type suffixes) when a requested document was not found. In the configuration, we defined the names for the error documents to be just their error number (without any suffix).

The names of the individual error documents are now determined like this (I'm using 403 as an example, think of it as a placeholder for any of the configured error documents):

- No file errordocs/403 should exist. Otherwise, it would be found and served (with the DefaultType, usually text/plain), all negotiation would be bypassed.
- For each language for which we have an internationalized version (note that this need not be the same set of languages for each error code - you can get by with a single language version until you actually *have* translated versions), a document errordocs/403.shtml.*lang* is created and filled with the error text in that language ([see below](#)).
- One fallback document called errordocs/403.shtml is created, usually by creating a symlink to the default language variant ([see below](#)).

The common header and footer files

By putting as much layout information in two special "include files", the error documents can be reduced to a bare minimum.

One of these layout files defines the HTML document header and a configurable list of paths to the icons to be shown in the resulting error document. These paths are exported as a set of XSSI environment variables and are later evaluated by the "footer" special file. The title of the current error (which is put into the TITLE tag and an H1 header) is simply passed in from the main error document in a variable called `title`.

By changing this file, the layout of all generated error messages can be changed in a second. (By exploiting the features of XSSI, you can easily define different layouts based on the current virtual host, or even based on the client's domain name).

The second layout file describes the footer to be displayed at the bottom of every error message. In this example, it shows an apache logo, the current server time, the server version string and adds a mail reference to the site's webmaster.

For simplicity, the header file is simply called `head.shtml` because it contains server-parsed content but no language specific information. The footer file exists once for each language translation, plus a symlink for the default language.

Example: for English, French and German versions (default english)

```

foot.shtml.en,
foot.shtml.fr,
foot.shtml.de,
foot.shtml symlink to foot.shtml.en

```

Both files are included into the error document by using the directives `<!--#include virtual="head" -->` and `<!--#include virtual="foot" -->` respectively: the rest of the magic occurs in `mod_negotiation` and in `mod_include`.

See [the listings below](#) to see an actual HTML implementation of the discussed example.

Creating ErrorDocuments in different languages

After all this preparation work, little remains to be said about the actual documents. They all share a simple common structure:

```
<!--#set var="title" value="error description title" -->
<!--#include virtual="head" -->
    explanatory error text
<!--#include virtual="foot" -->
```

In the [listings section](#), you can see an example of a [400 Bad Request] error document. Documents as simple as that certainly cause no problems to translate or expand.

The fallback language

Do we need a special handling for languages other than those we have translations for? We did set the LanguagePriority, didn't we?!

Well, the LanguagePriority directive is for the case where the client does not express any language priority at all. But what happens in the situation where the client wants one of the languages we do not have, and none of those we do have?

Without doing anything, the Apache server will usually return a [406 no acceptable variant] error, listing the choices from which the client may select. But we're in an error message already, and important error information might get lost when the client had to choose a language representation first.

So, in this situation it appears to be easier to define a fallback language (by copying or linking, *e.g.*, the english version to a language-less version). Because the negotiation algorithm prefers "more specialized" variants over "more generic" variants, these generic alternatives will only be chosen when the normal negotiation did not succeed.

A simple shell script to do it (execute within the errordocs/ dir):

```
for f in *.shtml.en
do
    ln -s $f `basename $f .en`
done
```

Customizing Proxy Error Messages

As of Apache-1.3, it is possible to use the ErrorDocument mechanism for proxy error messages as well (previous versions always returned fixed predefined error messages).

Most proxy errors return an error code of [500 Internal Server Error]. To find out whether a particular error document was invoked on behalf of a proxy error or because of some other server error, and what the reason for the failure was, you can check the contents of the new ERROR_NOTES CGI environment variable: if invoked for a proxy error, this variable will contain the actual proxy error message text in HTML form.

The following excerpt demonstrates how to exploit the ERROR_NOTES variable within an error document:

```
<!--#if expr="$REDIRECT_ERROR_NOTES = ''" -->
<p>
    The server encountered an unexpected condition
    which prevented it from fulfilling the request.
</p>
<p>
    <A HREF="mailto:<!--#echo var="SERVER_ADMIN" -->"
      SUBJECT="Error message [<!--#echo var="REDIRECT_STATUS" -->] <!--#echo var="title"
--> for <!--#echo var="REQUEST_URI" -->">
    Please forward this error screen to <!--#echo var="SERVER_NAME" -->'s
    WebMaster</A>; it includes useful debugging information about
    the Request which caused the error.
    <pre><!--#printenv --></pre>
</p>
<!--#else -->
    <!--#echo var="REDIRECT_ERROR_NOTES" -->
<!--#endif -->
```

HTML listing of the discussed example

So, to summarize our example, here's the complete listing of the 400.shtml.en document. You will notice that it contains almost nothing but the error text (with conditional additions). Starting with this example, you will find it easy to add more error documents, or to translate the error documents to different languages.

```

<!--#set var="title" value="Bad Request"
--><!--#include virtual="head" --><P>
  Your browser sent a request that this server could not understand:
  <BLOCKQUOTE>
    <STRONG><!--#echo var="REQUEST_URI" --></STRONG>
  </BLOCKQUOTE>
  The request could not be understood by the server due to malformed
  syntax. The client should not repeat the request without
  modifications.
</P>
<P>
<!--#if expr="$HTTP_REFERER != ''" -->
  Please inform the owner of
  <A HREF="<!--#echo var="HTTP_REFERER" -->">the referring page</A> about
  the malformed link.
<!--#else -->
  Please check your request for typing errors and retry.
<!--#endif -->
</P>
<!--#include virtual="foot" -->

```

Here is the complete head.shtml file (the funny line breaks avoid empty lines in the document after XSSI processing). Note the configuration section at top. That's where you configure the images and logos as well as the apache documentation directory. Look how this file displays two different logos depending on the content of the virtual host name (\$SERVER_NAME), and that an animated apache logo is shown if the browser appears to support it (the latter requires server configuration lines of the form
 BrowserMatch "^Mozilla/[2-4]" anigif
 for browser types which support animated GIFs).

```

<!--#if expr="$SERVER_NAME = /*.*\mycompany\.com/"
--><!--#set var="IMG_CorpLogo"
  value="http://$SERVER_NAME:$SERVER_PORT/errordocs/CorpLogo.gif"
--><!--#set var="ALT_CorpLogo" value="Powered by Linux!"
--><!--#else
--><!--#set var="IMG_CorpLogo"
  value="http://$SERVER_NAME:$SERVER_PORT/errordocs/PrivLogo.gif"
--><!--#set var="ALT_CorpLogo" value="Powered by Linux!"
--><!--#endif
--><!--#set var="IMG_BgImage"
value="http://$SERVER_NAME:$SERVER_PORT/errordocs/BgImage.gif"
--><!--#set var="DOC_Apache" value="http://$SERVER_NAME:$SERVER_PORT/Apache/"
--><!--#if expr="$anigif"
--><!--#set var="IMG_Apache"
value="http://$SERVER_NAME:$SERVER_PORT/icons/apache_anim.gif"
--><!--#else
--><!--#set var="IMG_Apache"
value="http://$SERVER_NAME:$SERVER_PORT/icons/apache_pb.gif"
--><!--#endif
--><!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML//EN">
<HTML>
<HEAD>
<TITLE>
  [<!--#echo var="REDIRECT_STATUS" -->] <!--#echo var="title" -->
</TITLE>
</HEAD>
<BODY BGCOLOR="white" BACKGROUND="<!--#echo var="IMG_BgImage" -->"><UL>
<H1 ALIGN="center">
  [<!--#echo var="REDIRECT_STATUS" -->] <!--#echo var="title" -->
  <IMG SRC="<!--#echo var="IMG_CorpLogo" -->"
    ALT="<!--#echo var="ALT_CorpLogo" -->" ALIGN=right>
</H1>
<HR><!-- ===== -->
</DIV>

```

and this is the foot.shtml.en file:

```

</DIV>
<HR>
<DIV ALIGN="right"><SMALL><SUP>Local Server time:
  <!--#echo var="DATE_LOCAL" -->

```

```
</SUP></SMALL></DIV>
<DIV ALIGN="center">
  <A HREF="<!--#echo var="DOC_Apache" -->">
  <IMG SRC="<!--#echo var="IMG_Apache" -->" BORDER=0 ALIGN="bottom"
    ALT="Powered by <!--#echo var="SERVER_SOFTWARE" -->"></A><BR>
  <SMALL><SUP><!--#set var="var"
    value="Powered by $SERVER_SOFTWARE -- File last modified on $LAST_MODIFIED"
    --><!--#echo var="var" --></SUP></SMALL>
</DIV>
<ADDRESS>If the indicated error looks like a misconfiguration, please inform
  <A HREF="mailto:<!--#echo var="SERVER_ADMIN" -->"
    SUBJECT="Feedback about Error message [<!--#echo var="REDIRECT_STATUS"
      -->] <!--#echo var="title" -->, req=<!--#echo var="REQUEST_URI" -->">
  <!--#echo var="SERVER_NAME" -->'s WebMaster</A>.
</ADDRESS>
</UL></BODY>
</HTML>
```

More welcome!

If you have tips to contribute, send mail to martin@apache.org

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Descriptors and Apache

A *descriptor*, also commonly called a *file handle* is an object that a program uses to read or write an open file, or open network socket, or a variety of other devices. It is represented by an integer, and you may be familiar with `stdin`, `stdout`, and `stderr` which are descriptors 0, 1, and 2 respectively. Apache needs a descriptor for each log file, plus one for each network socket that it listens on, plus a handful of others. Libraries that Apache uses may also require descriptors. Normal programs don't open up many descriptors at all, and so there are some latent problems that you may experience should you start running Apache with many descriptors (*i.e.*, with many virtual hosts).

The operating system enforces a limit on the number of descriptors that a program can have open at a time. There are typically three limits involved here. One is a kernel limitation, depending on your operating system you will either be able to tune the number of descriptors available to higher numbers (this is frequently called `FD_SETSIZE`). Or you may be stuck with a (relatively) low amount. The second limit is called the *hard resource* limit, and it is sometimes set by root in an obscure operating system file, but frequently is the same as the kernel limit. The third limit is called the *soft resource* limit. The soft limit is always less than or equal to the hard limit. For example, the hard limit may be 1024, but the soft limit only 64. Any user can raise their soft limit up to the hard limit. Root can raise the hard limit up to the system maximum limit. The soft limit is the actual limit that is used when enforcing the maximum number of files a process can have open.

To summarize:

```
#open files <= soft limit <= hard limit <= kernel limit
```

You control the hard and soft limits using the `limit` (csh) or `ulimit` (sh) directives. See the respective man pages for more information. For example you can probably use `ulimit -n unlimited` to raise your soft limit up to the hard limit. You should include this command in a shell script which starts your webserver.

Unfortunately, it's not always this simple. As mentioned above, you will probably run into some system limitations that will need to be worked around somehow. Work was done in version 1.2.1 to improve the situation somewhat. Here is a partial list of systems and workarounds (assuming you are using 1.2.1 or later):

BSDI 2.0

Under BSDI 2.0 you can build Apache to support more descriptors by adding `-DFD_SETSIZE=nnn` to `EXTRA_CFLAGS` (where `nnn` is the number of descriptors you wish to support, keep it less than the hard limit). But it will run into trouble if more than approximately 240 Listen directives are used. This may be cured by rebuilding your kernel with a higher `FD_SETSIZE`.

FreeBSD 2.2, BSDI 2.1+

Similar to the BSDI 2.0 case, you should define `FD_SETSIZE` and rebuild. But the extra Listen limitation doesn't exist.

Linux

By default Linux has a kernel maximum of 256 open descriptors per process. There are several patches available for the 2.0.x series which raise this to 1024 and beyond, and you can find them in the "unofficial patches" section of [the Linux Information HQ](#). None of these patches are perfect, and an entirely different approach is likely to be taken during the 2.1.x development. Applying these patches will raise the `FD_SETSIZE` used to compile all programs, and unless you rebuild all your libraries you should avoid running any other program with a soft descriptor limit above 256. As of this writing the patches available for increasing the number of descriptors do not take this into account. On a dedicated webserver you probably won't run into trouble.

Solaris through 2.5.1

Solaris has a kernel hard limit of 1024 (may be lower in earlier versions). But it has a limitation that files using the `stdio` library cannot have a descriptor above 255. Apache uses the `stdio` library for the `ErrorLog` directive. When you have more than approximately 110 virtual hosts (with an error log and an access log each) you will need to build Apache with

-DHIGH_SLACK_LINE=256 added to EXTRA_CFLAGS. You will be limited to approximately 240 error logs if you do this.

AIX

AIX version 3.2?? appears to have a hard limit of 128 descriptors. End of story. Version 4.1.5 has a hard limit of 2000.

SCO OpenServer

Edit the `/etc/conf/cf.d/stune` file or use `/etc/conf/cf.d/configure` choice 7 (User and Group configuration) and modify the `NOFILES` kernel parameter to a suitably higher value. SCO recommends a number between 60 and 11000, the default is 110. Relink and reboot, and the new number of descriptors will be available.

Compaq Tru64 UNIX/Digital UNIX/OSF

1. Raise `open_max_soft` and `open_max_hard` to 4096 in the proc subsystem. Do a man on `sysconfig`, `sysconfigdb`, and `sysconfigtab`.
2. Raise `max-vnodes` to a large number which is greater than the number of apache processes * 4096 (Setting it to 250,000 should be good for most people). Do a man on `sysconfig`, `sysconfigdb`, and `sysconfigtab`.
3. If you are using Tru64 5.0, 5.0A, or 5.1, define `NO_SLACK` to work around a bug in the OS.
`CFLAGS="-DNO_SLACK" ./configure`

Others

If you have details on another operating system, please submit it through our [Bug Report Page](#).

In addition to the problems described above there are problems with many libraries that Apache uses. The most common example is the `bind` DNS resolver library that is used by pretty much every unix, which fails if it ends up with a descriptor above 256. We suspect there are other libraries that similar limitations. So the code as of 1.2.1 takes a defensive stance and tries to save descriptors less than 16 for use while processing each request. This is called the *low slack line*.

Note that this shouldn't waste descriptors. If you really are pushing the limits and Apache can't get a descriptor above 16 when it wants it, it will settle for one below 16.

In extreme situations you may want to lower the low slack line, but you shouldn't ever need to. For example, lowering it can increase the limits 240 described above under Solaris and BSDI 2.0. But you'll play a delicate balancing game with the descriptors needed to serve a request. Should you want to play this game, the compile time parameter is `LOW_SLACK_LINE` and there's a tiny bit of documentation in the header file `httpd.h`.

Finally, if you suspect that all this slack stuff is causing you problems, you can disable it. Add `-DNO_SLACK` to `EXTRA_CFLAGS` and rebuild. But please report it to our [Bug Report Page](#) so that we can investigate.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Connections in the FIN_WAIT_2 state and Apache

1. What is the FIN_WAIT_2 state?

Starting with the Apache 1.2 betas, people are reporting many more connections in the FIN_WAIT_2 state (as reported by `netstat`) than they saw using older versions. When the server closes a TCP connection, it sends a packet with the FIN bit set to the client, which then responds with a packet with the ACK bit set. The client then sends a packet with the FIN bit set to the server, which responds with an ACK and the connection is closed. The state that the connection is in during the period between when the server gets the ACK from the client and the server gets the FIN from the client is known as FIN_WAIT_2. See the [TCP RFC](#) for the technical details of the state transitions.

The FIN_WAIT_2 state is somewhat unusual in that there is no timeout defined in the standard for it. This means that on many operating systems, a connection in the FIN_WAIT_2 state will stay around until the system is rebooted. If the system does not have a timeout and too many FIN_WAIT_2 connections build up, it can fill up the space allocated for storing information about the connections and crash the kernel. The connections in FIN_WAIT_2 do not tie up an httpd process.

2. But why does it happen?

There are numerous reasons for it happening, some of them may not yet be fully clear. What is known follows.

Buggy clients and persistent connections

Several clients have a bug which pops up when dealing with [persistent connections](#) (aka keepalives). When the connection is idle and the server closes the connection (based on the [KeepAliveTimeout](#)), the client is programmed so that the client does not send back a FIN and ACK to the server. This means that the connection stays in the FIN_WAIT_2 state until one of the following happens:

- The client opens a new connection to the same or a different site, which causes it to fully close the older connection on that socket.
- The user exits the client, which on some (most?) clients causes the OS to fully shutdown the connection.
- The FIN_WAIT_2 times out, on servers that have a timeout for this state.

If you are lucky, this means that the buggy client will fully close the connection and release the resources on your server. However, there are some cases where the socket is never fully closed, such as a dialup client disconnecting from their provider before closing the client. In addition, a client might sit idle for days without making another connection, and thus may hold its end of the socket open for days even though it has no further use for it. **This is a bug in the browser or in its operating system's TCP implementation.**

The clients on which this problem has been verified to exist:

- Mozilla/3.01 (X11; I; FreeBSD 2.1.5-RELEASE i386)
- Mozilla/2.02 (X11; I; FreeBSD 2.1.5-RELEASE i386)
- Mozilla/3.01Gold (X11; I; SunOS 5.5 sun4m)
- MSIE 3.01 on the Macintosh
- MSIE 3.01 on Windows 95

This does not appear to be a problem on:

- Mozilla/3.01 (Win95; I)

It is expected that many other clients have the same problem. What a client **should do** is periodically check its open socket(s) to see if they have been closed by the server, and close their side of the connection if the server has closed. This check need only occur once every few seconds, and may even be detected by a OS signal on some systems (*e.g.*, Win95 and NT clients have this capability, but they seem to be ignoring it).

Apache **cannot** avoid these FIN_WAIT_2 states unless it disables persistent connections for the buggy clients, just like we recommend doing for Navigator 2.x clients due to other bugs. However, non-persistent connections increase the total number of connections needed per client and slow retrieval of an image-laden web page. Since non-persistent connections have their own resource consumptions and a short waiting period after each closure, a busy server may need persistence in order to best serve its clients.

As far as we know, the client-caused FIN_WAIT_2 problem is present for all servers that support persistent connections, including Apache 1.1.x and 1.2.

A necessary bit of code introduced in 1.2

While the above bug is a problem, it is not the whole problem. Some users have observed no FIN_WAIT_2 problems with Apache 1.1.x, but with 1.2b enough connections build up in the FIN_WAIT_2 state to crash their server. The most likely source for additional FIN_WAIT_2 states is a function called `lingering_close()` which was added between 1.1 and 1.2. This function is necessary for the proper handling of persistent connections and any request which includes content in the message body (*e.g.*, PUTs and POSTs). What it does is read any data sent by the client for a certain time after the server closes the connection. The exact reasons for doing this are somewhat complicated, but involve what happens if the client is making a request at the same time the server sends a response and closes the connection. Without lingering, the client might be forced to reset its TCP input buffer before it has a chance to read the server's response, and thus understand why the connection has closed. See the [appendix](#) for more details.

The code in `lingering_close()` appears to cause problems for a number of factors, including the change in traffic patterns that it causes. The code has been thoroughly reviewed and we are not aware of any bugs in it. It is possible that there is some problem in the BSD TCP stack, aside from the lack of a timeout for the FIN_WAIT_2 state, exposed by the `lingering_close` code that causes the observed problems.

3. What can I do about it? There are several possible workarounds to the problem, some of which work better than others.

Add a timeout for FIN_WAIT_2

The obvious workaround is to simply have a timeout for the FIN_WAIT_2 state. This is not specified by the RFC, and could be claimed to be a violation of the RFC, but it is widely recognized as being necessary. The following systems are known to have a timeout:

- [FreeBSD](#) versions starting at 2.0 or possibly earlier.
- [NetBSD](#) version 1.2(?)
- [OpenBSD](#) all versions(?)
- [BSD/OS](#) 2.1, with the [K210-027](#) patch installed.
- [Solaris](#) as of around version 2.2. The timeout can be tuned by using `ndd` to modify `tcp_fin_wait_2_flush_interval`, but the default should be appropriate for most servers and improper tuning can have negative impacts.
- [Linux](#) 2.0.x and earlier(?)
- [HP-UX](#) 10.x defaults to terminating connections in the FIN_WAIT_2 state after the normal keepalive timeouts. This does not refer to the persistent connection or HTTP keepalive timeouts, but the `SO_LINGER` socket option which is enabled by Apache. This parameter can be adjusted by using `net tune` to modify parameters such as `tcp_keepstart` and `tcp_keepstop`. In later revisions, there is an explicit timer for connections in FIN_WAIT_2 that can be modified; contact HP support for details.
- [SGI IRIX](#) can be patched to support a timeout. For IRIX 5.3, 6.2, and 6.3, use patches 1654, 1703 and 1778 respectively. If you have trouble locating these patches, please contact your SGI support channel for help.
- [NCR's MP RAS Unix](#) 2.xx and 3.xx both have FIN_WAIT_2 timeouts. In 2.xx it is non-tunable at 600 seconds,

while in 3.xx it defaults to 600 seconds and is calculated based on the tunable "max keep alive probes" (default of 8) multiplied by the "keep alive interval" (default 75 seconds).

- [Sequent's ptx/TCP/IP for DYNIX/ptx](#) has had a FIN_WAIT_2 timeout since around release 4.1 in mid-1994.

The following systems are known to not have a timeout:

- [SunOS 4.x](#) does not and almost certainly never will have one because it is at the very end of its development cycle for Sun. If you have kernel source should be easy to patch.

There is a [patch available](#) for adding a timeout to the FIN_WAIT_2 state; it was originally intended for BSD/OS, but should be adaptable to most systems using BSD networking code. You need kernel source code to be able to use it. If you do adapt it to work for any other systems, please drop me a note at marc@apache.org.

Compile without using `lingering_close()`

It is possible to compile Apache 1.2 without using the `lingering_close()` function. This will result in that section of code being similar to that which was in 1.1. If you do this, be aware that it can cause problems with PUTs, POSTs and persistent connections, especially if the client uses pipelining. That said, it is no worse than on 1.1, and we understand that keeping your server running is quite important.

To compile without the `lingering_close()` function, add `-DNO_LINGCLOSE` to the end of the `EXTRA_CFLAGS` line in your Configuration file, rerun `Configure` and rebuild the server.

Use `SO_LINGER` as an alternative to `lingering_close()`

On most systems, there is an option called `SO_LINGER` that can be set with `setsockopt(2)`. It does something very similar to `lingering_close()`, except that it is broken on many systems so that it causes far more problems than `lingering_close`. On some systems, it could possibly work better so it may be worth a try if you have no other alternatives.

To try it, add `-DUSE_SO_LINGER -DNO_LINGCLOSE` to the end of the `EXTRA_CFLAGS` line in your Configuration file, rerun `Configure` and rebuild the server.

NOTE: Attempting to use `SO_LINGER` and `lingering_close()` at the same time is very likely to do very bad things, so don't.

Increase the amount of memory used for storing connection state

BSD based networking code:

BSD stores network data, such as connection states, in something called an mbuf. When you get so many connections that the kernel does not have enough mbufs to put them all in, your kernel will likely crash. You can reduce the effects of the problem by increasing the number of mbufs that are available; this will not prevent the problem, it will just make the server go longer before crashing.

The exact way to increase them may depend on your OS; look for some reference to the number of "mbufs" or "mbuf clusters". On many systems, this can be done by adding the line `NMBCLUSTERS="n"`, where n is the number of mbuf clusters you want to your kernel config file and rebuilding your kernel.

Disable KeepAlive

If you are unable to do any of the above then you should, as a last resort, disable KeepAlive. Edit your `httpd.conf` and change "KeepAlive On" to "KeepAlive Off".

4. Feedback If you have any information to add to this page, please contact me at marc@apache.org.
5. Appendix

Below is a message from Roy Fielding, one of the authors of HTTP/1.1.

Why the lingering close functionality is necessary with HTTP

The need for a server to linger on a socket after a close is noted a couple times in the HTTP specs, but not explained. This explanation is based on discussions between myself, Henrik Frystyk, Robert S. Thau, Dave Raggett, and John C. Mallery in the hallways of MIT while I was at W3C.

If a server closes the input side of the connection while the client is sending data (or is planning to send data), then the server's TCP stack will signal an RST (reset) back to the client. Upon receipt of the RST, the client will flush its own incoming TCP buffer back to the un-ACKed packet indicated by the RST packet argument. If the server has sent a message, usually an error response, to the client just before the close, and the client receives the RST packet before its application code has read the error message from its incoming TCP buffer and before the server has received the ACK sent by the client upon receipt of that buffer, then the RST will flush the error message before the client application has a chance to see it. The result is that the client is left thinking that the connection failed for no apparent reason.

There are two conditions under which this is likely to occur:

1. sending POST or PUT data without proper authorization
2. sending multiple requests before each response (pipelining) and one of the middle requests resulting in an error or other break-the-connection result.

The solution in all cases is to send the response, close only the write half of the connection (what shutdown is supposed to do), and continue reading on the socket until it is either closed by the client (signifying it has finally read the response) or a timeout occurs. That is what the kernel is supposed to do if SO_LINGER is set. Unfortunately, SO_LINGER has no effect on some systems; on some other systems, it does not have its own timeout and thus the TCP memory segments just pile-up until the next reboot (planned or not).

Please note that simply removing the linger code will not solve the problem -- it only moves it to a different and much harder one to detect.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Known Problems in Clients

Over time the Apache Group has discovered or been notified of problems with various clients which we have had to work around, or explain. This document describes these problems and the workarounds available. It's not arranged in any particular order. Some familiarity with the standards is assumed, but not necessary.

For brevity, *Navigator* will refer to Netscape's Navigator product (which in later versions was renamed "Communicator" and various other names), and *MSIE* will refer to Microsoft's Internet Explorer product. All trademarks and copyrights belong to their respective companies. We welcome input from the various client authors to correct inconsistencies in this paper, or to provide us with exact version numbers where things are broken/fixes.

For reference, [RFC1945](#) defines HTTP/1.0, and [RFC2068](#) defines HTTP/1.1. Apache as of version 1.2 is an HTTP/1.1 server (with an optional HTTP/1.0 proxy).

Various of these workarounds are triggered by environment variables. The admin typically controls which are set, and for which clients, by using [mod_browser](#). Unless otherwise noted all of these workarounds exist in versions 1.2 and later.

Trailing CRLF on POSTs

This is a legacy issue. The CERN webserver required POST data to have an extra CRLF following it. Thus many clients send an extra CRLF that is not included in the Content-Length of the request. Apache works around this problem by eating any empty lines which appear before a request.

Broken keepalive

Various clients have had broken implementations of *keepalive* (persistent connections). In particular the Windows versions of Navigator 2.0 get very confused when the server times out an idle connection. The workaround is present in the default config files:

```
BrowserMatch Mozilla/2 nokeepalive
```

Note that this matches some earlier versions of MSIE, which began the practice of calling themselves *Mozilla* in their user-agent strings just like Navigator.

MSIE 4.0b2, which claims to support HTTP/1.1, does not properly support keepalive when it is used on 301 or 302 (redirect) responses. Unfortunately Apache's `nokeepalive` code prior to 1.2.2 would not work with HTTP/1.1 clients. You must apply [this patch](#) to version 1.2.1. Then add this to your config:

```
BrowserMatch "MSIE 4\.0b2;" nokeepalive
```

Incorrect interpretation of HTTP/1.1 in response

To quote from section 3.1 of RFC1945:

HTTP uses a "<MAJOR>.<MINOR>" numbering scheme to indicate versions of the protocol. The protocol versioning policy is intended to allow the sender to indicate the format of a message and its capacity for understanding further HTTP communication, rather than the features obtained via that communication.

Since Apache is an HTTP/1.1 server, it indicates so as part of its response. Many client authors mistakenly treat this part of the response as an indication of the protocol that the response is in, and then refuse to accept the response.

The first major indication of this problem was with AOL's proxy servers. When Apache 1.2 went into beta it was the first wide-spread HTTP/1.1 server. After some discussion, AOL fixed their proxies. In anticipation of similar problems, the

`force-response-1.0` environment variable was added to Apache. When present Apache will indicate "HTTP/1.0" in response to an HTTP/1.0 client, but will not in any other way change the response.

The pre-1.1 Java Development Kit (JDK) that is used in many clients (including Navigator 3.x and MSIE 3.x) exhibits this problem. As do some of the early pre-releases of the 1.1 JDK. We think it is fixed in the 1.1 JDK release. In any event the workaround:

```
BrowserMatch Java/1.0 force-response-1.0
BrowserMatch JDK/1.0 force-response-1.0
```

RealPlayer 4.0 from Progressive Networks also exhibits this problem. However they have fixed it in version 4.01 of the player, but version 4.01 uses the same `User-Agent` as version 4.0. The workaround is still:

```
BrowserMatch "RealPlayer 4.0" force-response-1.0
```

Requests use HTTP/1.1 but responses must be in HTTP/1.0

MSIE 4.0b2 has this problem. Its Java VM makes requests in HTTP/1.1 format but the responses must be in HTTP/1.0 format (in particular, it does not understand *chunked* responses). The workaround is to fool Apache into believing the request came in HTTP/1.0 format.

```
BrowserMatch "MSIE 4\.0b2;" downgrade-1.0 force-response-1.0
```

This workaround is available in 1.2.2, and in a [patch](#) against 1.2.1.

Boundary problems with header parsing

All versions of Navigator from 2.0 through 4.0b2 (and possibly later) have a problem if the trailing CRLF of the response header starts at offset 256, 257 or 258 of the response. A `BrowserMatch` for this would match on nearly every hit, so the workaround is enabled automatically on all responses. The workaround implemented detects when this condition would occur in a response and adds extra padding to the header to push the trailing CRLF past offset 258 of the response.

Multipart responses and Quoted Boundary Strings

On multipart responses some clients will not accept quotes (") around the boundary string. The MIME standard recommends that such quotes be used. But the clients were probably written based on one of the examples in RFC2068, which does not include quotes. Apache does not include quotes on its boundary strings to workaround this problem.

Byterange requests

A byterange request is used when the client wishes to retrieve a portion of an object, not necessarily the entire object. There was a very old draft which included these byteranges in the URL. Old clients such as Navigator 2.0b1 and MSIE 3.0 for the MAC exhibit this behavior, and it will appear in the servers' access logs as (failed) attempts to retrieve a URL with a trailing ";xxx-yyy". Apache does not attempt to implement this at all.

A subsequent draft of this standard defines a header `Request-Range`, and a response type `multipart/x-byteranges`. The HTTP/1.1 standard includes this draft with a few fixes, and it defines the header `Range` and type `multipart/byteranges`.

Navigator (versions 2 and 3) sends both `Range` and `Request-Range` headers (with the same value), but does not accept a `multipart/byteranges` response. The response must be `multipart/x-byteranges`. As a workaround, if Apache receives a `Request-Range` header it considers it "higher priority" than a `Range` header and in response uses `multipart/x-byteranges`.

The Adobe Acrobat Reader plugin makes extensive use of byteranges and prior to version 3.01 supports only the `multipart/x-byterange` response. Unfortunately there is no clue that it is the plugin making the request. If the plugin is used with Navigator, the above workaround works fine. But if the plugin is used with MSIE 3 (on Windows) the workaround won't work because MSIE 3 doesn't give the `Range-Request` clue that Navigator does. To workaround this, Apache special cases "MSIE 3" in the `User-Agent` and serves `multipart/x-byteranges`. Note that the necessity for this with MSIE 3 is actually due to the Acrobat plugin, not due to the browser.

Netscape Communicator appears to not issue the non-standard `Request-Range` header. When an Acrobat plugin prior to version 3.01 is used with it, it will not properly understand byteranges. The user must upgrade their Acrobat reader to 3.01.

Set-Cookie header is unmergeable

The HTTP specifications say that it is legal to merge headers with duplicate names into one (separated by commas). Some browsers that support Cookies don't like merged headers and prefer that each `Set-Cookie` header is sent separately. When parsing the headers returned by a CGI, Apache will explicitly avoid merging any `Set-Cookie` headers.

Expires headers and GIF89A animations

Navigator versions 2 through 4 will erroneously re-request GIF89A animations on each loop of the animation if the first response included an `Expires` header. This happens regardless of how far in the future the expiry time is set. There is no workaround supplied with Apache, however there are hacks for [1.2](#) and for [1.3](#).

POST without Content-Length

In certain situations Navigator 3.01 through 3.03 appear to incorrectly issue a POST without the request body. There is no known workaround. It has been fixed in Navigator 3.04, Netscapes provides some [information](#). There's also [some information](#) about the actual problem.

JDK 1.2 betas lose parts of responses.

The http client in the JDK1.2beta2 and beta3 will throw away the first part of the response body when both the headers and the first part of the body are sent in the same network packet AND keep-alive's are being used. If either condition is not met then it works fine.

See also Bug-ID's 4124329 and 4125538 at the java developer connection.

If you are seeing this bug yourself, you can add the following BrowserMatch directive to work around it:

```
BrowserMatch "Java1\.2beta[23]" nokeepalive
```

We don't advocate this though since bending over backwards for beta software is usually not a good idea; ideally it gets fixed, new betas or a final release comes out, and no one uses the broken old software anymore. In theory.

Content-Type change is not noticed after reload

Navigator (all versions?) will cache the `content-type` for an object "forever". Using reload or shift-reload will not cause Navigator to notice a `content-type` change. The only work-around is for the user to flush their caches (memory and disk). By way of an example, some folks may be using an old `mime.types` file which does not map `.htm` to `text/html`, in this case Apache will default to sending `text/plain`. If the user requests the page and it is served as `text/plain`. After the admin fixes the server, the user will have to flush their caches before the object will be shown with the correct `text/html` type.

MSIE Cookie problem with expiry date in the year 2000

MSIE versions 3.00 and 3.02 (without the Y2K patch) do not handle cookie expiry dates in the year 2000 properly. Years after 2000 and before 2000 work fine. This is fixed in IE4.01 service pack 1, and in the Y2K patch for IE3.02. Users should avoid using expiry dates in the year 2000.

Lynx incorrectly asking for transparent content negotiation

The Lynx browser versions 2.7 and 2.8 send a "negotiate: trans" header in their requests, which is an indication the browser supports transparent content negotiation (TCN). However the browser does not support TCN. As of version 1.3.4, Apache supports TCN, and this causes problems with these versions of Lynx. As a workaround future versions of Apache will ignore this header when sent by the Lynx client.

MSIE 4.0 mishandles Vary response header

MSIE 4.0 does not handle a Vary header properly. The Vary header is generated by mod_rewrite in apache 1.3. The result is an error from MSIE saying it cannot download the requested file. There are more details in [PR#4118](#).

A workaround is to add the following to your server's configuration files:

```
BrowserMatch "MSIE 4\.0" force-no-vary
```

(This workaround is only available with releases **after** 1.3.6 of the Apache Web server.)

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Why We Took PEM Out of Apache

On May 17th, 1995, we were asked by a representative of NCSA to remove any copies of NCSA httpd prior to 1.4.1 from our web site. They were mandated by the NSA to inform us that redistribution of pre-1.4.1 code violated the same laws that make distributing Phill Zimmerman's PGP package to other countries illegal. There was **no** encryption in NCSA's httpd, only hooks to publicly available libraries of PEM code. By the NSA's rules, even hooks to this type of application is illegal.

Because Apache is based on NCSA code, and we had basically not touched that part of the software, we were informed that Apache was also illegal to distribute to foreign countries, and advised (not mandated) by NCSA to remove it. So, we removed both the copies of the NCSA httpd we had, and all versions of Apache previous to 0.6.5.

The Apache members are strong advocates of the right to digital privacy, so the decision to submit to the NSA and remove the code was not an easy one. Here are some elements in our rationale:

- The PEM code in httpd was not widely used. No major site relied upon its use, so its loss is not a blow to encryption and security on the world wide web. There are other efforts designed to give much more flexible security - SSL and SHTTP - so this wasn't a function whose absence would really be missed on a functional level.
- We didn't feel like being just a couple more martyrs in a fight being fought very well by many other people. Rather than have the machine that supports the project confiscated or relocated to South Africa, *etc.*, we think there are more efficient methods to address the issue.

It kind of sickens us that we had to do it, but so be it.

Patches that re-implement the PEM code may be available at a foreign site soon. If it does show up, we'll point to it - that can't be illegal!

Finally, here is a compendium of pointers to sites related to encryption and export law. We can't promise this list will be up to date, so send us mail when you see a problem or want a link added. Thanks.

- [Yahoo - Science: Mathematics: Security and Encryption](#)
- [EFF Crypto/Privacy/Security Archive](#)
- [Crypto page at Quadralay](#)
- [Cryptography Export Control Archives \(Cygnus\)](#)
- [ICLU - Your Rights in Cyberspace](#)

Brian, brian@hyperreal.com

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Configuring Multiple IP Addresses

This material is originally from John Ioannidis (ji@polaris.ctr.columbia.edu) I have condensed it some and applied some corrections for SunOS 4.1.x courtesy of Chuck Smoko (csmoko@relay.nswc.navy.mil).

Bob Baggerman (bob@bizweb.com)
12 Jan 94

=====
John Ionnidis writes:

This is a topic that comes up once in a while on comp.protocols.tcp-ip and other newsgroups. The question is, how to get a machine with one network interface to respond to more than one IP addresses.

I have a solution than might suit you. For my doctoral work (there's a paper about it in this year's ('91) SIGCOMM, also available for anonymous FTP from cs.columbia.edu:/pub/ji/sigcomm*.ps.Z), I've developed what I call the "Virtual Interface" (VIF). To the networking code, it looks like an interface. It gets ifattach()ed when you open the /dev/vif* device, and then you can ifconfig it as you like. It does not have an if_input procedure; it only has an if_output. Packets that it receives (from higher-level protocols) which have its IP address, it simply loops back (like any well-behaved if driver). Packets that it receives that are destined for some other address, it encapsulates in an encapsulation protocol I call IPIP (IP-within-IP, protocol number IPPROTO_IPIP == 94), and sends it to another machine that groks that encapsulation protocol. This feature you won't need, but here's how to have multiple IP addresses on a machine with a single real interface:

Let's say your primary interface's IP address is 198.3.2.1, and you also want it to respond to addresses 198.4.3.2 and 198.5.4.3 (note that these are three distinct class C addresses in three distinct class C nets). Here are the ifconfigs:

```
ifconfig le0 198.3.2.1 up -trailers      # config primary interface

ifconfig vif0 198.4.3.2 up              # config first virtual interface
route delete net 198.4.3 198.4.3.2     # delete spurious route
route add host 198.4.3.2 198.4.3.2 0   # add route for this i/f

ifconfig vif1 198.5.4.3 up              # config second virtual interface
route delete net 198.5.4 198.5.4.3     # delete spurious route
route add host 198.5.4.3 198.5.4.3 0   # add route for this i/f
```

The route deletes are needed because the ifconfig creates a default route to the interface's network, which can cause problems; all that's needed is the (host) route to the interface's address.

Configuring Multiple IP Addresses

Now, get le0's ethernet address (say, 8:0:20:3:2:1), and add the following static ARP entries:

```
arp -s 198.4.3.2 8:0:20:3:2:1 pub
arp -s 198.5.4.3 8:0:20:3:2:1 pub
```

This will cause any ARP requests for the VIF addresses to be replied with your machine's ethernet address.

Now, make sure your default route is to your segment's gateway, through the real interface. Finally, make sure your routers and/or hosts on the same segment as yours know that 198.4.3.2 and 198.5.4.3 are on that cable.

Here's what you've accomplished.

ARP requests for any of your host's addresses will be replied to with the host's ethernet address (the real one, because that's what it is, the virtual ones because of the public static arp entries). Packets reaching your host with any of these addresses will be accepted by the ip_input routine because they match the address of one of the host's interfaces. Packets leaving your host can have any of its addresses (real and virtual).

The code for vif follows. To use it, put the stuff in netinet/if_vif.c and netinet/if_vif.h, configure your kernel with the number of virtual interfaces you want using a line like:

```
pseudo-device    vif4            # Virtual IP interface
```

in your configuration file, and the line

```
netinet/if_vif.c    optional vif device-driver
```

in the "files" file. Also, add the appropriate entries in conf.c, so that you can access the if_attach() routine when you open the device:

```
----- conf.c-----
```

add this in the appropriate place in the headers of conf.c:

```
-----
#include "vif.h"
#if NVIF > 0
int    vifopen(), vifclose(), vifread(), vifwrite(), vifselect(), vifioctl();
#else
#define vifopen        nodev
#define vifclose       nodev
#define vifread        nodev
#define vifwrite       nodev
#define vifselect      nodev
#define vifioctl       nodev
#endif
-----
```

then, way down in the definition for cdevsw[]:

```
-----
    vifopen,        vifclose,        vifread,        vifwrite,        /*14*/
    vifioctl,      nodev,            nodev,          0,
    0,             nodev,
-----
```

 Make sure you remember the correct major device number, 14 in this case!

Finally, here's the code. It has the tunneling pieces removed (you need more code to use that anyway), and it comes from a Mach 2.6 kernel; it should compile on any Berkeley-derived unix with minor changes (most likely only in the includes).

```
-----netinet/if_vif.h-----
typedef struct
{
    struct ifnet    vif_if;
    struct ifnet    *vif_sif;    /* slave interface */
    int             vif_flags;
} vif_softc_t;

#define VIFMTU    (1024+512)
-----

and

-----netinet/if_vif.c-----
/*
 * Virtual IP interface module.
 */

#include "param.h"
#include "../sys/system.h"
#include "../sys/mbuf.h"
#include "../sys/socket.h"
#include "../sys/errno.h"
#include "../sys/ioctl.h"

#include "../net/if.h"
#include "../net/netisr.h"
#include "../net/route.h"

#ifdef INET
#include "../netinet/in.h"
#include "../netinet/in_system.h"
#include "../netinet/in_var.h"
#include "../netinet/ip.h"
#endif

#include "in_pcb.h"
#include "vif.h"

typedef struct
{
    struct ifnet    vif_if;
    struct ifnet    *vif_sif;    /* slave interface */
    int             vif_flags;
} vif_softc_t;

#define VIFMTU    (1024+512)

vif_softc_t vif_softc[NVIF];

int vifs_initd = 0;
```

```

vifattach()
{
    register int i;
    register struct ifnet *ifp;
    int      vifoutput(), vififioctl();

    for (i=0; i<NVIF; i++)
    {
        ifp = &vif_softc[i].vif_if;
        ifp->if_name = "vif";
        ifp->if_unit = i;
        ifp->if_mtu = VIFMTU;
        ifp->if_flags = IFF_LOOPBACK | IFF_NOARP;
        ifp->if_ioctl = vififioctl;
        ifp->if_output = vifoutput;
        if_attach(ifp);
    }
}

vifopen(dev, flag)
int dev, flag;
{
    int unit;

    if (!vifs_init)
    {
        vifattach();
        vifs_init = 1;
        printf("vif initialized\n");
    }

    unit = minor(dev);
    if ((unit < 0) || (unit >= NVIF))
    {
        return ENXIO;
    }

    return 0;
}

vifclose(dev, flag)
int dev, flag;
{
    return 0;
}

vifread()
{
    return ENXIO;
}

vifwrite()
{
    return ENXIO;
}

vifselect()
{
    return ENXIO;
}

```

```

vifoutput(ifp, m0, dst)
    struct ifnet *ifp;
    register struct mbuf *m0;
    struct sockaddr *dst;
{
    int s;
    register struct ifqueue *ifq;
    struct mbuf *m;
    struct sockaddr_in *din;

    if (dst->sa_family != AF_INET)
    {
        printf("%s%d: can't handle af%d\n",
            ifp->if_name, ifp->if_unit,
            dst->sa_family);
        m_freem(m0);
        return (EAFNOSUPPORT);
    }

    din = (struct sockaddr_in *)dst;

    if (din->sin_addr.s_addr == IA_SIN(ifp->if_addrlist)->sin_addr.s_addr)
    {
        /* printf("%s%d: looping\n", ifp->if_name, ifp->if_unit); */

        /*
         * Place interface pointer before the data
         * for the receiving protocol.
         */
        if (m0->m_off <= MMAXOFF &&
            m0->m_off >= MMINOFF + sizeof(struct ifnet *)) {
            m0->m_off -= sizeof(struct ifnet *);
            m0->m_len += sizeof(struct ifnet *);
        } else {
            MGET(m, M_DONTWAIT, MT_HEADER);
            if (m == (struct mbuf *)0)
                return (ENOBUFS);
            m->m_off = MMINOFF;
            m->m_len = sizeof(struct ifnet *);
            m->m_next = m0;
            m0 = m;
        }
        *(mtod(m0, struct ifnet **)) = ifp;
        s = splimp();
        ifp->if_opackets++;
        ifq = &ipintrq;
        if (IF_QFULL(ifq)) {
            IF_DROP(ifq);
            m_freem(m0);
            splx(s);
            return (ENOBUFS);
        }
        IF_ENQUEUE(ifq, m0);
        schednetisr(NETISR_IP);
        ifp->if_ipackets++;
        splx(s);
        return (0);
    }

    return EHOSTUNREACH;
}

```

```

/*
 * Process an ioctl request.
 */
/* ARGSUSED */
vififiocctl(ifp, cmd, data)
    register struct ifnet *ifp;
    int cmd;
    caddr_t data;
{
    int error = 0;

    switch (cmd) {

    case SIOCSIFADDR:
        ifp->if_flags |= IFF_UP;
        /*
         * Everything else is done at a higher level.
         */
        break;

    default:
        error = EINVAL;
    }
    return (error);
}

vifiocctl(dev, cmd, arg, mode)
dev_t dev;
int cmd;
caddr_t arg;
int mode;
{
    int unit;

    unit = minor(dev);
    if ((unit < 0) || (unit >= NVIF))
        return ENXIO;

    return EINVAL;
}
-----

```

To use it, compile your kernel, and reboot. Then create the vif device:

```
# mknod /dev/vif c 14 0
```

(or whatever major number it ended up being), and echo something into it:

```
# echo > /dev/vif
```

This will cause the device to be opened, which will `if_attach` the interfaces. If you feel like playing with the code, you may want to `kmem_alloc()` the `vif_softc` structure at open time, and use the minor number of the device to tell it how many interfaces to create.

Now you can go ahead and `ifconfig` etc.

I'll be happy to answer minor questions, and hear about success and failure stories, but I cannot help you if you don't already know how

to hack kernels.

Good luck!

/ji

In-Real-Life: John "Heldenprogrammer" Ioannidis

E-Mail-To: ji@cs.columbia.edu

V-Mail-To: +1 212 854 8120

P-Mail-To: 450 Computer Science \n Columbia University \n New York, NY 10027

Note: there is also a [commercial-product-turned-freeware called "Col. Patch"](#) which does this as a loadable kernel module for SunOS 4.1.3_U1.

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

HTTP/1.1 KeepAlive problems with Netscape 3.0

Date: Mon, 1 Jul 1996 16:03:06 -0700 (PDT)
From: Alexei Kosut <akosut@organic.com>
To: Apache Group
Subject: Re: keepalive and windoze

Good news and good news (of a sort)..

I was able to snag a Windows 95 machine here at Organic, and tried out some things:

1) On Netscape 3.0b4, I was able to reproduce the bug, each and every time. It's really simple: go to the Network Setup panel. Set it to "Connect Every Time" and only let it have 1 connection at once (this may not be necessary, but it's helpful). Then load an image that's kept-alive. Then wait until the connection times out (this depends on the server - 10-30 seconds, except for MIIS, which never times out, near as I can tell). Then hit reload. It will hang. (actually, once it crashed).

2) This happens with all forms of server. Apache 1.1, Netscape 2.0, Spyglass 1.2, NCSA 1.5 (although, as stated, I couldn't test MIIS).

3) Netscape 3.0b5 does, indeed, *not* have this bug. At least, I couldn't get it to perform such. Yipee.

So, we just put up a note on the web page. Make sure we say that all the servers have the bug, it's a Windows bug, and Netscape Navigator 3.0b5 works around it. That way, no one can yell at us. Yes?

-- Alexei Kosut <akosut@organic.com>
<http://www.nueva.pvt.k12.ca.us/~akosut/>

The Apache HTTP Server
<http://www.apache.org/>

Apache HTTP Server Version 1.3





Apache HTTP Server Version 1.3

Reading Client Input in Apache 1.2

Apache 1.1 and earlier let modules handle POST and PUT requests by themselves. The module would, on its own, determine whether the request had an entity, how many bytes it was, and then called a function (`read_client_block`) to get the data.

However, HTTP/1.1 requires several things of POST and PUT request handlers that did not fit into this module, and all existing modules have to be rewritten. The API calls for handling this have been further abstracted, so that future HTTP protocol changes can be accomplished while remaining backwards-compatible.

The New API Functions

```
int ap_setup_client_block (request_rec *, int read_policy);
int ap_should_client_block (request_rec *);
long ap_get_client_block (request_rec *, char *buffer, int buffer_size);
```

1. Call `ap_setup_client_block()` near the beginning of the request handler. This will set up all the necessary properties, and will return either OK, or an error code. If the latter, the module should return that error code. The second parameter selects the policy to apply if the request message indicates a body, and how a chunked transfer-coding should be interpreted. Choose one of

<code>REQUEST_NO_BODY</code>	Send 413 error if message has any body
<code>REQUEST_CHUNKED_ERROR</code>	Send 411 error if body without Content-Length
<code>REQUEST_CHUNKED_DECHUNK</code>	If chunked, remove the chunks for me.
<code>REQUEST_CHUNKED_PASS</code>	Pass the chunks to me without removal.

In order to use the last two options, the caller **MUST** provide a buffer large enough to hold a chunk-size line, including any extensions.

2. When you are ready to possibly accept input, call `ap_should_client_block()`. This will tell the module whether or not to read input. If it is 0, the module should assume that the input is of a non-entity type (*e.g.*, a GET request). A nonzero response indicates that the module should proceed (to step 3). This step also sends a 100 Continue response to HTTP/1.1 clients, so should not be called until the module is ***definitely*** ready to read content. (otherwise, the point of the 100 response is defeated). Never call this function more than once.
3. Finally, call `ap_get_client_block` in a loop. Pass it a buffer and its size. It will put data into the buffer (not necessarily the full buffer, in the case of chunked inputs), and return the length of the input block. When it is done reading, it will return 0 if EOF, or -1 if there was an error.

As an example, please look at the code in `mod_cgi.c`. This is properly written to the new API guidelines.

Apache HTTP Server Version 1.3

