Apache HTTP Server Version 2.0

Site Map

Apache HTTP Server Version 2.0

INDEX

Apache HTTP Server Version 2.0

Search

Maintained by the Apache HTTP Server Documentation Project.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Upgrading to 2.0 from 1.3

In order to assist folks upgrading, we maintain a document describing information critical to existing Apache users. 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

- Apache now uses an `autoconf` and `libtool` system for configuring the build processes. Using this system is similar to, but not the same as, using the APACI system in Apache 1.3.
- In addition to the usual selection of modules which you can choose to compile, Apache 2.0 has moved the main part of request processing into Multi-Processing Modules (MPMs).

Run-Time Configuration Changes

- Many directives that were in the core server in Apache 1.3 are now in the MPMs. If you wish the behavior of the server to be as similar as possible to the behavior of Apache 1.3, you should select the prefork MPM. Other MPMs will have different directives to control process creation and request processing.
- The proxy module has been revamped to bring it up to HTTP/1.1. Among the important changes, proxy access control is now placed inside a <Proxy> block rather than a <Directory proxy:> block.
- The handling of PATH_INFO (trailing path information after the true filename) has changed for some modules. Modules that were previously implemented as a handler but are now implemented as a filter may no longer accept requests with PATH_INFO. Filters such as INCLUDES are implemented on top of the core handler, and therefore reject requests with PATH_INFO. You can use the AcceptPathInfo directive to force the core handler to accept requests with PATH_INFO and thereby restore the ability to use PATH_INFO in server-side includes.
- The `CacheNegotiatedDocs` directive now takes the argument `on` or `off`. Existing instances of `CacheNegotiatedDocs` should be replaced with `CacheNegotiatedDocs on`.
- The `ErrorDocument` directive no longer uses a quote at the beginning of the argument to indicate a text message. Instead, you should enclose the message in double quotes. For example, existing instances of

  ```
  ErrorDocument 403 "Some Message
  ```

  should be replaced with

  ```
  ErrorDocument 403 "Some Message"
  ```

  As long as the second argument is not a valid URL or pathname, it will be treated as a text message.
- The `AccessConfig` and `ResourceConfig` directives no longer exist. Existing instances of these directives can be replaced with the `Include` directive which has equivalent functionality. If you were making use of the default values of these directives without including them in the configuration files, you may need to add `Include conf/access.conf` and `Include conf/srm.conf` to your httpd.conf. In order to assure that Apache reads the configuration files in the same order as was implied by the older directives, the `Include` directives should be placed at the end of httpd.conf, with the one for `srm.conf` preceding the one for `access.conf`.
- The `BindAddress` and `Port` directives no longer exist. Equivalent functionality is provided with the more flexible `Listen` directive.
- Another use of the `Port` directive in Apache-1.3 was setting the port number to be used in self-referential URL's. The Apache-2.0 equivalent is the new `ServerName` syntax: it has been changed to allow specifying both the hostname *and* the port number for self-referential URL's in one directive.
- The `ServerType` directive no longer exists. The method used to serve requests is now determined by the selection of MPM. There is currently no MPM designed to be launched by inetd.
- The mod_log_agent and mod_log_referer modules which provided the `AgentLog`, `RefererLog` and `RefererIgnore` directives have been removed. Agent and referer logs are still available using the CustomLog directive of mod_log_config.
- The `AddModule` and `ClearModuleList` directives no longer exist. These directives where used to ensure that modules could be

enabled in the correct order. The new Apache 2.0 API allows modules to explicitly specify their ordering, eliminating the need for these directives.

- The `FancyIndexing` directive has been removed. The same functionality is available through the `FancyIndexing` option to the IndexOptions directive.

Misc Changes

- The `httpd` command line option `-S` which was used for printing the virtual host configuration has been replaced by `-t -D DUMP_VHOSTS`.
- The module mod_auth_digest, which was experimental in Apache 1.3 is now a standard module.
- The mod_mmap_static module, which was experimental in Apache 1.3 has been replaced with mod_file_cache.
- The distribution has been completely reorganized so that it no longer contains an independent `src` directory. Instead, the sources are logically organized under the main distribution directory, and installations of the compiled server should be directed to a separate directory.

Third Party Modules

Extensive changes were made to the server API in Apache 2.0. Existing modules designed for the Apache 1.3 API will **not** work in Apache 2.0 without modification. Details are provided in the developer documentation.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Overview of New Features in Apache 2.0

Enhancements: Core | Module

Core Enhancements:

Unix Threading

On Unix systems with POSIX threads support, Apache can now run in a hybrid multiprocess, multithreaded mode. This improves scalability for many, but not all configurations.

New Build System

The build system has been rewritten from scratch to be based on autoconf and libtool. This makes Apache's configuration system more similar to that of other packages.

Multiprotocol Support

Apache now has some of the infrastructure in place to support serving multiple protocols. mod_echo has been written as an example.

Better support for non-Unix platforms

Apache 2.0 is faster and more stable on non-Unix platforms such as BeOS, OS/2, and Windows. With the introduction of platform-specific multi-processing modules (MPMs) and the Apache Portable Runtime (APR), these platforms are now implemented in their native API, avoiding the often buggy and poorly performing POSIX-emulation layers.

New Apache API

The API for modules has changed significantly for 2.0. Many of the module-ordering/-priority problems from 1.3 should be gone. 2.0 does much of this automatically, and module ordering is now done per-hook to allow more flexibility. Also, new calls have been added that provide additional module capabilities without patching the core Apache server.

IPv6 Support

On systems where IPv6 is supported by the underlying Apache Portable Runtime library, Apache gets IPv6 listening sockets by default. Additionally, the Listen, NameVirtualHost, and <VirtualHost> directives support IPv6 numeric address strings (e.g., "Listen [fe80::1]:8080").

Filtering

Apache modules may now be written as filters which act on the stream of content as it is delivered to or from the server. This allows, for example, the output of CGI scripts to be parsed for Server Side Include directives using the INCLUDES filter in mod_include.

Multilanguage Error Responses

Error response messages to the browser are now provided in several languages, using SSI documents. They may be customized by the administrator to achieve a consistent look and feel.

Simplified configuration

Many confusing directives have been simplified. The often confusing Port and BindAddress directives are gone; only the Listen directive is used for IP address binding; the ServerName directive specifies the server name and port number only for redirection and vhost recognition.

Native Windows NT Unicode Support

Apache 2.0 on Windows NT now uses utf-8 for all filename encodings. These directly translate to the underlying Unicode file system, providing multilanguage support for all Windows NT-based installations, including Windows 2000 and Windows XP. *This support does not extend to Windows 95, 98 or ME, which continue to use the machine's local codepage for filesystem access.*

Module Enhancements:

mod_ssl

New module in Apache 2.0. This module is an interface to the SSL/TLS encryption protocols provided by OpenSSL.

mod_dav

New module in Apache 2.0. This module implements the HTTP Distributed Authoring and Versioning (DAV) specification for posting and maintaining web content.

mod_auth_digest

Includes additional support for session caching across processes using shared memory.

mod_charset_lite

New module in Apache 2.0. This experimental module allows for character set translation or recoding.

mod_file_cache

New module in Apache 2.0. This module includes the functionality of mod_mmap_static in Apache 1.3, plus adds further caching abilities.

mod_headers

This module is much more flexible in Apache 2.0. It can now modify request headers used by mod_proxy, and it can conditionally set response headers.

mod_proxy

The proxy module has been completely rewritten to take advantage of the new filter infrastructure and to implement a more reliable, HTTP/1.1 compliant proxy. In addition, new <Proxy> configuration sections provide more readable (and internally faster) control of proxied sites; overloaded <Directory "proxy:..."> configuration are not supported. The module is now divided into specific protocol support modules including proxy_connect, proxy_ftp and proxy_http.

mod_negotiation

A new ForceLanguagePriority directive can be used to assure that the client receives a single document in all cases, rather than NOT ACCEPTABLE or MULTIPLE CHOICES responses. In addition, the negotiation and MultiViews algorithms have been cleaned up to provide more consistent results and a new form of type map that can include document content is provided.

mod_autoindex

Autoindex'ed directory listings can now be configured to use HTML tables for cleaner formatting, and allow finer-grained control of sorting, including version-sorting, and wildcard filtering of the directory listing.

mod_include

New directives allow the default start and end tags for SSI elements to be changed and allow for error and time format configuration to take place in the main configuration file rather than in the SSI document. Results from regular expression parsing and grouping (now based on Perl's regular expression syntax) can be retrieved using mod_include's variables $0 .. $9.

mod_auth_dbm

Now supports multiple types of DBM-like databases using the AuthDBMType directive.

mod_auth_db

Has been removed in favor of mod_auth_dbm with the AuthDBMType directive.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Compiling and Installing

This document covers compilation and installation of Apache on Unix and Unix-like systems only. For compiling and installation on Windows, see Using Apache with Microsoft Windows. For other platforms, see the platform documentation.

Apache 2.0's configuration and installation environment has changed completely from Apache 1.3. Apache 1.3 used a custom set of scripts to achieve easy installation. Apache 2.0 now uses libtool and autoconf to create an environment that looks like many other Open Source projects.

Overview for the impatient

Download	`$ lynx http://www.apache.org/dist/httpd/httpd-2_0_NN.tar.gz`

Extract	`$ gzip -d httpd-2_0_NN.tar.gz`
	`$ tar xvf httpd-2_0_NN.tar`

Configure	`$ ./configure --prefix=PREFIX`
Compile	`$ make`
Install	`$ make install`
Customize	`$ vi PREFIX/conf/httpd.conf`
Test	`$ PREFIX/bin/apachectl start`

NN must be replaced with the current minor version number, and *PREFIX* must be replaced with the filesystem path under which the server should be installed. If *PREFIX* is not specified, it defaults to `/usr/local/apache2`.

Each section of the compilation and installation process is described in more detail below, beginning with the requirements for compiling and installing Apache HTTPD.

Requirements

The following requirements exist for building Apache:

- Disk Space

  Make sure you have at least 50 MB of temporary free disk space available. After installation Apache occupies approximately 10 MB of disk space. The actual disk space requirements will vary considerably based on your chosen configuration options and any third-party modules.

- ANSI-C Compiler and Build System

  Make sure you have an ANSI-C compiler installed. The GNU C compiler (GCC) from the Free Software Foundation (FSF) is recommended (version 2.7.2 is fine). If you don't have GCC then at least make sure your vendor's compiler is ANSI compliant. In addition, your `PATH` must contain basic build tools such as `make`.

- Accurate time keeping

  Elements of the HTTP protocol are expressed as the time of day. So, it's time to investigate setting some time synchronization facility on your system. Usually the ntpdate or xntpd programs are used for this purpose which are based on the Network Time Protocol (NTP). See the Usenet newsgroup comp.protocols.time.ntp and the NTP homepage for more details about NTP software and public time servers.

- Perl 5 [OPTIONAL]

  For some of the support scripts like apxs or dbmmanage (which are written in Perl) the Perl 5 interpreter is required (versions 5.003 and 5.004 are fine). If no such interpreter is found by the `configure' script there is no harm. Of course, you still can build and install Apache 2.0. Only those support scripts cannot be used. If you have multiple Perl interpreters installed (perhaps a Perl 4 from the vendor and a Perl 5 from your own), then it is recommended to use the --with-perl option (see below) to make sure the correct one is selected by ./configure.

Download

Apache can be downloaded from the Apache Software Foundation download site or from a nearby mirror.

Version numbers that end in `alpha` indicate early pre-test versions which may or may not work. Version numbers ending in `beta` indicate more reliable releases that still require further testing or bug fixing. If you wish to download the best available production release of the Apache HTTP Server, you should choose the latest version with neither `alpha` nor `beta` in its filename.

After downloading, especially if a mirror site is used, it is important to verify that you have a complete and unmodified version of the Apache HTTP Server. This can be accomplished by testing the downloaded tarball against the PGP signature. This, in turn, is a two step procedure. First, you must obtain the `KEYS` file from the Apache distribution site. (To assure that the `KEYS` file itself has not been modified, it may be a good idea to use a file from a previous distribution of Apache or import the keys from a public key server.) The keys are imported into your personal key ring using one of the following commands (depending on your pgp version):

```
$ pgp < KEYS
```

or

```
$ gpg --import KEYS
```

The next step is to test the tarball against the PGP signature, which should always be obtained from the main Apache website. The signature file has a filename identical to the source tarball with the addition of `.asc`. Then you can check the distribution with one of the following commands (again, depending on your pgp version):

```
$ pgp httpd-2_0_NN.tar.gz.asc
```

or

```
$ gpg --verify httpd-2_0_NN.tar.gz.asc
```

You should receive a message like

```
Good signature from user "Martin Kraemer <martin@apache.org>".
```

Depending on the trust relationships contained in your key ring, you may also receive a message saying that the relationship between the key and the signer of the key cannot be verified. This is not a problem if you trust the authenticity of the `KEYS` file.

Extract

Extracting the source from the Apache HTTPD tarball is a simple matter of uncompressing, and then untarring:

```
$ gzip -d httpd-2_0_NN.tar.gz
$ tar xvf httpd-2_0_NN.tar
```

This will create a new directory under the current directory containing the source code for the distribution. You should `cd` into that directory before proceeding with compiling the server.

Configuring the source tree

The next step is to configure the Apache source tree for your particular platform and personal requirements. This is done using the script `configure` included in the root directory of the distribution. (Developers downloading the CVS version of the Apache source tree will need to have `autoconf` and `libtool` installed and will need to run `buildconf` before proceeding with the next steps. This is not necessary for official releases.)

To configure the source tree using all the default options, simply type `./configure`. To change the default options, `configure` accepts a variety of variables and command line options. Environment variables are generally placed before the `./configure` command, while other options are placed after. The most important option here is the location prefix where Apache is to be installed later, because Apache has to be configured for this location to work correctly. But there are a lot of other options available for your pleasure.

For a short impression of what possibilities you have, here is a typical example which compiles Apache for the installation tree /sw/pkg/apache with a particular compiler and flags plus the two additional modules mod_rewrite and mod_speling for later loading through the DSO mechanism:

```
$ CC="pgcc" CFLAGS="-O2" \
./configure --prefix=/sw/pkg/apache \
--enable-rewrite=shared \
--enable-speling=shared
```

When configure is run it will take several minutes to test for the availability of features on your system and build Makefiles which will later be used to compile the server.

The easiest way to find all of the configuration flags for Apache is to run ./configure --help. What follows is a brief description of most of the arguments and environment variables.

Environment Variables

The autoconf build process uses several environment variables to configure the build environment. In general, these variables change the method used to build Apache, but not the eventual features of the server. These variables can be placed in the environment before invoking `configure`, but it is usually easier to specify them on the `configure` command line as demonstrated in the example above.

`CC=...`

The name of the C compiler command.

`CPPFLAGS=...`

Miscellaneous C preprocessor and compiler options.

`CFLAGS=...`

Debugging and optimization options for the C compiler.

`LDFLAGS=...`

Miscellaneous options to be passed to the linker.

`LIBS=...`

Library location information ("-L" and "-l" options) to pass to the linker.

`INCLUDES=...`

Header file search directories ("-I*dir*").

`TARGET=...` [Default: apache]

Name of the executable which will be built.

`NOTEST_CPPFLAGS=...`

`NOTEST_CFLAGS=...`

`NOTEST_LDFLAGS=...`

`NOTEST_LIBS=...`

These variables share the same function as their non-NOTEST namesakes. However, the variables are applied to the build process only after autoconf has performed its feature testing. This allows the inclusion of flags which will cause problems during feature testing, but must be used for the final compilation.

`SHLIB_PATH=...`

Options which specify shared library paths for the compiler and linker.

autoconf Output Options

`--help`

Prints the usage message including all available options, but does not actually configure anything.

`--quiet`

Prevents the printing of the usual "checking..." messages.

`--verbose`

Prints much more information during the configuration process, including the names of all the files examined.

Pathnames

There are currently two ways to configure the pathnames under which Apache will install its files. First, you can specify a directory and have Apache install itself under that directory in its default locations.

`--prefix=`*`PREFIX`* [Default: /usr/local/apache2]

Specifies the directory under which the Apache files will be installed.

It is possible to specify that architecture-dependent files should be placed under a different directory.

`--exec-prefix=`*`EPREFIX`* [Default: *PREFIX*]

Specifies the directory under which architecture-dependent files will be placed.

The second, and more flexible way to configure the install path locations for Apache is using the `config.layout` file. Using this method, it is possible to separately specify the location for each type of file within the Apache installation. The `config.layout` file contains several example configurations, and you can also create your own custom configuration following the examples. The different layouts in this file are grouped into `<Layout FOO>...</Layout>` sections and referred to by name as in `FOO`.

`--enable-layout=`*`LAYOUT`*

Use the named layout in the `config.layout` file to specify the installation paths.

Presently it is not possible to mix the `--enable-layout` and `--prefix` options. Nor is it possible to individually specify detailed pathnames on the `configure` command line. If you want just a basic install, you can simply use the `--prefix` option on its own. If you want to customize your install, you should edit the `config.layout` file and use the `--enable-layout` option.

Modules

Apache is a modular server. Only the most basic functionality is included in the core server. Extended features are available in various modules. During the configuration process, you must select which modules to compile for use with your server. You can view a list of modules included in the documentation. Those modules with a status of "Base" are included by default and must be specifically disabled if you do not want them. Modules with any other status must be specifically enabled if you wish to use them.

There are two ways for a module to be compiled and used with Apache. Modules may be *statically compiled*, which means that they are permanently included in the Apache binary. Alternatively, if your operating system supports Dynamic Shared Objects (DSOs) and autoconf can detect that support, then modules may be *dynamically compiled*. DSO modules are stored separately from the Apache binary, and may be included or excluded from the server using the run-time configuration directives provided by mod_so. The mod_so is automatically included in the server if any dynamic modules are included in the compilation. If you would like to make your server capable of loading DSOs without actually compiling any dynamic modules, you can explicitly `--enable-so`.

`--enable-`*`MODULE`*`[=shared]`

Compile and include the module *MODULE*. The identifier *MODULE* is the Module Identifier from the module documentation without the "_module" string. To compile the module as a DSO, add the option `=shared`.

`--disable-`*`MODULE`*

Remove the module *MODULE* which would otherwise be compiled and included.

`--enable-modules=`*`MODULE-LIST`*

Compile and include the modules listed in the space-separated *MODULE-LIST*.

`--enable-mods-shared=`*`MODULE-LIST`*

Compile and include the modules in the space-separated *MODULE-LIST* as dynamically loadable (DSO) modules.

The *MODULE-LIST* in the `--enable-modules` and `--enable-mods-shared` options is usually a space-separated list of module identifiers. For example, to enable mod_dav and mod_info, you can either use

```
./configure --enable-dav --enable-info
```

or, equivalently,

```
./configure --enable-modules="dav info"
```

In addition, the special keywords `all` or `most` can be used to add all or most of the modules in one step. You can then remove any modules that you do not want with the `--disable-`*`MODULE`* option. For example, to include all modules as DSOs with the exception of mod_info, you can use

```
./configure --enable-mods-shared=all --disable-info
```

In addition to the standard set of modules, Apache 2.0 also includes a choice of Multi-Processing Modules (MPMs). One, and only one MPM must be included in the compilation process. The default MPMs for each platform are listed on the MPM documentation page, but can be overridden on the `configure` command line.

`--with-mpm=`*`NAME`*

Choose the mpm *NAME*.

Suexec

Apache includes a support program called suexec which can be used to isolate user CGI programs. However, if suexec is improperly configured, it can cause serious security problems. Therefore, you should carefully read and consider the suexec documentation before implementing this feature.

Build

Now you can build the various parts which form the Apache package by simply running the command:

```
$ make
```

Please be patient here, since a base configuration takes approximately 3 minutes to compile under a Pentium III/Linux 2.2 system, but this will vary widely depending on your hardware and the number of modules which you have enabled.

Install

Now its time to install the package under the configured installation *PREFIX* (see `--prefix` option above) by running:

```
$ make install
```

If you are upgrading, the installation will not overwrite your configuration files or documents.

Customize

Next, you can customize your Apache HTTP server by editing the configuration files under *PREFIX*/conf/.

```
$ vi PREFIX/conf/httpd.conf
```

Have a look at the Apache manual under docs/manual/ or http://httpd.apache.org/docs/ for a complete reference of available configuration directives.

Test

Now you can start your Apache HTTP server by immediately running:

```
$ PREFIX/bin/apachectl start
```

and then you should be able to request your first document via URL http://localhost/. The web page you see is located under the DocumentRoot which will usually be *`PREFIX`*`/htdocs/`. Then stop the server again by running:

```
$ PREFIX/bin/apachectl stop
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Starting Apache

Starting Apache On Windows

On Windows, Apache is normally run as a service on Windows NT, or as a console application on Windows 95. 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.

If the Listen specified in the configuration file is 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 performed 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 the selected Multi-Processing Module.

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
- 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 Version 2.0

Apache HTTP Server Version 2.0

Stopping and Restarting the Server

This document covers stopping and restarting Apache on Unix-like systems. 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.

A shell script called apachectl is provided which automates the processing of signalling Apache. For details about this script, see the documentation on starting Apache.

Stop Now

Signal: TERM
`apachectl stop`

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.

Graceful Restart

Signal: USR1
`apachectl graceful`

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.

On certain platforms that do not allow USR1 to be used for a graceful restart, an alternative signal may be used (such as WINCH). apachectl graceful will send the right signal for your platform.

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.

Restart Now

Signal: HUP
`apachectl restart`

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.

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 Version 2.0

Apache HTTP Server Version 2.0

Configuration Files

Main Configuration Files

Related Modules	Related Directives
mod_mime	<IfDefine> Include 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. 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 Version 2.0

Apache HTTP Server Version 2.0

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.

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 Version 2.0

Apache HTTP Server Version 2.0

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

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.

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 Version 2.0

INDEX

Apache HTTP Server Version 2.0

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 occuring. 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

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 rotatelogs for this purpose. For example, to rotate the logs every 24 hours, you can use:

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

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 mod_rewrite	PidFile 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 Version 2.0

Apache HTTP Server Version 2.0

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.

Related Modules	Related Directives
mod_alias mod_rewrite mod_userdir mod_speling mod_vhost_alias	Alias AliasMatch CheckSpelling DocumentRoot 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 Version 2.0

INDEX

Apache HTTP Server Version 2.0

Security Tips for Server Configuration

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 desginated 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
</Directory>
```

This prevents the use of `.htaccess` files in all directories apart from those specifically enabled.

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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Dynamic Shared Object (DSO) Support

The Apache HTTP Server is a modular program where the administrator can choose the functionality to include in the server by selecting a set of modules. The modules can be statically compiled into the `httpd` binary when the server is built. Alternatively, modules can be compiled as Dynamic Shared Objects (DSOs) that exist separately from the main `httpd` binary file. DSO modules may be compiled at the time the server is built, or they may be compiled and added at a later time using the Apache Extension Tool (apxs).

This document describes how to use DSO modules as well as the theory behind their use.

Related Modules	Related Directives
mod_so	LoadModule

Implementation

The DSO support for loading individual Apache modules is based on a module named `mod_so.c` which must be statically compiled into the Apache core. It is the only module besides `core.c` which cannot be put into a DSO itself. 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-`*`module`*`=shared` option as disucussed in the install documentation. 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.

Usage Summary

To give you an overview of the DSO features of Apache 2.0, here is a short and concise summary:

1. Build and install a *distributed* Apache module, say `mod_foo.c`, into its own DSO `mod_foo.so`:

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

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

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

3. Configure Apache for *later installation* of shared modules:

```
$ ./configure --enable-so
$ make install
```

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 using apxs:

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

In all cases, once the shared module is compiled, you must use a `LoadModule` directive in `httpd.conf` to tell Apache to activate the module.

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.

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.

Advantages and Disadvantages

The above DSO based features 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 `configure` options 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 or loading the code yourself via `dlopen()`.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

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 is 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.

Type map files should have the same name as the resource which they are describing, and 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 shown below. This file would be named `foo.html`, as it describes a resource named `foo`.

```
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
```

Note also that a typemap file will take precedence over the filename's extension, even when Multiviews is on. 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 in bytes. 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.

Using a type map file is preferred over `MultiViews` because it requires less CPU time, and less file access, to parse a file explicitly listing the various resource variants, than to have to look at every matching file, and parse its file extensions.

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 the 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
foo.html.en	foo foo.html	-
foo.en.html	foo	foo.html
foo.html.en.gz	foo foo.html	foo.gz foo.html.gz
foo.en.html.gz	foo	foo.html foo.html.gz foo.gz
foo.gz.html.en	foo foo.gz foo.gz.html	foo.html
foo.html.gz.en	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 rsp. 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 Version 2.0

Apache HTTP Server Version 2.0

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 Version 2.0

Apache HTTP Server Version 2.0

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. However, it needs to be told to listen to specific ports, 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.

The `Listen` directive tells the server to accept incoming requests only on the specified port or address-and-port combinations. If only a port number is specified in the `Listen` directive, the server listens to the given port on all interfaces. 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
```

IPv6 addresses must be surrounded in square brackets, as in the following example:

```
   Listen [fe80::a00:20ff:fea7:ccea]:80
```

Special IPv6 considerations

When APR supports IPv6, Apache will create IPv6-capable listening sockets by default (i.e., when no IP address is specified on the Listen directive). In other words, when APR supports IPv6,

```
   Listen 80
```

is equivalent to

```
   Listen [::]:80
```

When APR does not support IPv6,

```
   Listen 80
```

is equivalent to

```
   Listen 0.0.0.0:80
```

On some platforms, such as NetBSD, binding to the IPv6 wildcard address ("::") does not allow Apache to accept connections on IPv4 interfaces. In this situation, multiple Listen directives are required, as shown below:

```
   Listen 0.0.0.0:80
   Listen [::]:80
```

Apache does not currently detect this, so the Listen statements must be edited manually by the administrator.

How this works with Virtual Hosts

Listen does not implement Virtual Hosts. It only tells the main server what addresses and ports to listen to. 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 first be told to listen to the address and port to be used. Then a <VirtualHost> section should be created for a 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 Listen directive, Virtual Hosts, DNS Issues and <VirtualHost> section.

Apache HTTP Server Version 2.0

INDEX

Apache HTTP Server Version 2.0

Apache Multi-Processing Modules

The Apache HTTP Server is designed to be a powerful and flexible web server that can work on a very wide variety of platforms in a range of different environments. Different platforms and different environments often require different features, or may have different ways of implementing the same feature most efficiently. Apache has always accommodated a wide variety of environments through its modular design. This design allows the webmaster to choose which features will be included in the server by selecting which modules to load either at compile-time or at run-time.

Apache 2.0 extends this modular design to the most basic functions of a web server. The server ships with a selection of Multi-Processing Modules (MPMs) which are responsible for binding to network ports on the machine, accepting requests, and dispatching children to handle the requests.

Extending the modular design to this level of the server allows two important benefits:

- Apache can more cleanly and efficiently support a wide variety of operating systems. In particular, the Windows version of Apache is now much more efficient, since mpm_winnt can use native networking features in place of the POSIX layer used in Apache 1.3. This benefit also extends to other operating systems that implement specialized MPMs.
- The server can be better customized for the needs of the particular site. For example, sites that need a great deal of scalability can choose to use a threaded MPM like worker, while sites requiring stability or compatibility with older software can use a preforking MPM. In addition, special features like serving different hosts under different userids (perchild) can be provided.

At the user level, MPMs appear much like other Apache modules. The main difference is that one and only one MPM must be loaded into the server at any time. The list of available MPMs appears on the module index page.

Choosing an MPM

MPMs must be chosen during configuration, and compiled into the server. Compilers are capable of optimizing a lot of functions if threads are used, but only if they know that threads are being used. Because some MPMs use threads on Unix and others don't, Apache will always perform better if the MPM is chosen at configuration time and built into Apache.

To actually choose the desired MPM, use the argument --with-mpm= *NAME* with the ./configure script. *NAME* is the name of the desired MPM.

Once the server has been compiled, it is possible to determine which MPM was chosen by using `./httpd -l`. This command will list every module that is compiled into the server, including the MPM.

MPM Defaults

- BeOS: beos
- OS/2: mpmt_os2
- Unix: prefork
- Windows: winnt

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

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

Related Modules	Related Directives
mod_env mod_rewrite mod_setenvif mod_unique_id	BrowserMatch BrowserMatchNoCase PassEnv 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 mod_cgi mod_headers mod_include mod_log_config mod_rewrite	Allow CustomLog Deny Header LogFormat 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 `INCLUDES` filter can print environment variables using the `echo` element, and can use environment variables in flow control elements to makes 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.

Conditional Response Headers

The `Header` directive can use the presence or absence of an environment variable to determine whether or not a certain HTTP header will be placed in the response to the client. This allows, for example, a certain response header to be sent only if a corresponding header is received in the request from the client.

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.

redirect-carefully

This forces the server to be more careful when sending a redirect to the client. This is typically used when a client has a known problem handling redirects. This was originally implemented as a result of a problem with Microsoft's WebFolders software which has a problem handling redirects on directory resources via DAV methods.

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>
```

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

Apache HTTP Server Version 2.0

INDEX

Apache HTTP Server Version 2.0

Apache's Handler Use

What is a Handler

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

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-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 Version 2.0

Apache HTTP Server Version 2.0

Filters

Related Modules	Related Directives
mod_defalte mod_ext_filter mod_include	AddInputFilter AddOutputFilter ExtFilterDefine ExtFilterOptions SetInputFilter SetOutputFilter

A *filter* is a process that is applied to data that is sent or received by the server. Data sent by clients to the server is processed by *input filters* while data sent by the server to the client is processed by *output filters*. Multiple filters can be applied to the data, and the order of the filters can be explicitly specified.

Filters are used internally by Apache to perform functions such as chunking and byte-range request handling. In addition, modules can provide filters that are selectable using run-time configuration directives. The set of filters that apply to data can be manipulated with the `SetInputFilter`, `SetOutputFilter`, `AddInputFilter`, and `AddOutputFilter` directives.

The following user-selectable filters are currently provided with the Apache HTTP Server distribution.

INCLUDES

Server-Side Includes processing by mod_include

DEFLATE

Compress output before sending it to the client using mod_deflate

In addition, the module mod_ext_filter allows for external programs to be defined as filters.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache suEXEC Support

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 exists, 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. 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-safepath=`*`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 Version 2.0

Apache HTTP Server Version 2.0

Warning: This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

Apache Performance Notes

Author: Dean Gaudet

Related Modules	Related Directives
mod_dir Multi-Processing module mod_status	AllowOverride DirectoryIndex HostnameLookups KeepAliveTimeout MaxSpareServers MinSpareServers Options (FollowSymLinks and FollowIfOwnerMatch) StartServers

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 webservers 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.

Also note that explicitly creating a `type-map` file provides better performance than using `MultiViews`, as the necessary information can be determined by reading this single file, rather than having to scan the directory for files.

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.

`USE_FLOCK_SERIALIZED_ACCEPT`

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

`USE_FCNTL_SERIALIZED_ACCEPT`

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

`USE_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).

`USE_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 `-DUSE_USLOCK_SERIALIZED_ACCEPT` on the `EXTRA_CFLAGS`.

`USE_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).

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" behaviour 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 behaviour 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 here */
            }
        }

        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 behaviour 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_TCP1, [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"..., 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_TCP1, [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 `writev` which combines both the headers and the first hunk of data:

```
    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"..., 6144}], 2) = 6389
```

When doing HTTP/1.1 chunked encoding Apache will generate up to four element `writev`s. 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_TCP1, [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
    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"..., 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 `mmapped` 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 2.0

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. Independed 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 is seems trivial it is actually trivial with mod_rewrite, only. Because the typical old mechanisms of URL *Aliases* (as provides 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 is tries to fix it themself in most of the cases, but sometimes this mechanism need 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 a 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 *independed*! What we want is to give the WWW namespace a consistent server-independend 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. The 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 aksed 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://newserver/~user/anypath`.

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

Structured Homedirs

Description:

Some sites with thousend 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 to 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 wanted 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 neccessary 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 rewritemap 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-Dependend Rewriting

Description:

When tricks like time-dependend 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-dependend 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 necesarry 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 permutated/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 webservers.

2. **DNS Load-Balancing**

   A sophisticated DNS-based method for load-balancing is to use the program `lbnamed` which can be found at http://www.stanford.edu/~schemers/docs/lbnamed/lbnamed.html. It is a Perl 5 program in conjunction with auxilliary 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
Listen               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 /apache-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 reponses
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_STRINGS) 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 `wwwidx` (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 `swwidx` program from within `/u/user/foo/` via hyperlink to

```
/internal/cgi/user/swwidx?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/wwwidx?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/wwwidx?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 seemless 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 invokation 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 a 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   ^(/[uge]/[^/]+/?.*):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, $mtime) = stat($file);
    return $mtime;
}

$mtimeL = &mystat($QS_f);
$mtime = $mtime;
for ($n = 0; $n &lt; $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_URL}  !^/commonurl1/.*
RewriteCond   %{REQUEST_URL}  !^/commonurl2/.*
    :
RewriteCond   %{REQUEST_URL}  !^/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 -
bsdti1.sdm.de  -
192.76.162.40  -
```

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-dependend deny...

```
RewriteCond %{REMOTE_HOST} ^badhost\.mydomain\.com$
RewriteRule !^http://[^/.]\.mydomain.com.*  - [F]
```

...and this one for a user@host-dependend 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       prg:/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 2.0

Apache HTTP Server Version 2.0

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 `-t -D DUMP_VHOSTS` 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 2.0

Apache HTTP Server Version 2.0

Name-based Virtual Host Support

This document describes when and how to use name-based virtual hosts.

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`. 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.

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 overriden 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:

```
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 2.0

Apache HTTP Server Version 2.0

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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Dynamically configured mass virtual hosting

This document describes how to efficiently serve an arbitrary number of virtual hosts with Apache 1.3.

Contents:

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, mome 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 2.0

Apache HTTP Server Version 2.0

Virtual Host examples for common setups

Base configuration

Additional features

Simple name-based vhosting

- **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:

  ```
  ...
  Listen 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:

  ```
  ...
  Listen 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:

  ```
  ...
  Listen 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 without the NameVirtualHost name:port or you try to use the Listen 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:

```
...
Listen 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:

  ```
  ...
  Listen 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:

  ```
  ...
  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 be used for a couple of name-based vhosts and the other addresses for IP-based vhosts.

  Server configuration:

```
...
Listen 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:

  ```
  ...
  Listen 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.

- **Setup:**

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

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 Listen, 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 `Listen` 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 Listen directive has two functions. One function is to determine the default network port Apache will bind to. 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 `Listen` 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. The IP address can also be the wildcard `*`.

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 `Listen` 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 `Listen` 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.

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 `Listen` by default. A wildcard port can be specified (*i.e.*, `_default_:*`) to catch requests to any available port. 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 2.0

Apache HTTP Server Version 2.0

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 you 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 2.0

Apache HTTP Server Version 2.0

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 webservers.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Frequently Asked Questions

The latest version of this FAQ is always available from the main Apache web site, at <http://httpd.apache.org/docs-2.0/faq/>. In addition, you can view this FAQ all in one page for easy searching and printing.

Since Apache 2.0 is very new, we don't yet know what the *Frequently Asked Questions* will be. While this section fills up, you should also consult the Apache 1.3 FAQ to see if your question is answered there.

Categories

Support

What do I do when I have problems?

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Frequently Asked Questions

Support

"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/apache2/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 forum**

   Apache has an active community of users who are willing to share their knowledge. Participating in this community is usually the best and fastest way to get answers to your questions and problems.

   Users mailing list

   USENET newsgroups:
   - comp.infosystems.www.servers.unix [nntp] [google]
   - comp.infosystems.www.servers.ms-windows [nntp] [google]
   - comp.infosystems.www.authoring.cgi [news] [google]
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`.)

Whom do I contact for support?

With several million users and fewer than forty volunteer developers, we cannot provide personal support for Apache. For free support, we suggest participating in a user forum.

Professional, commercial support for Apache is available from a number of companies.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

SSL/TLS Strong Encryption

The Apache HTTP Server module mod_ssl provides an interface to the OpenSSL library, which provides Strong Encryption using the Secure Sockets Layer and Transport Layer Security protocols. The module and this documentation are based on Ralf S. Engelschall's mod_ssl project.

Extensive documentation on the directives and environment variables provided by this module is provided in the mod_ssl reference documentation.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

SSL/TLS Strong Encryption: An Introduction

``The nice thing about standards is that there are so many to choose from. And if you really don't like all the standards you just have to wait another year until the one arises you are looking for.''
A. Tanenbaum, ``Introduction to Computer Networks''

As an introduction this chapter is aimed at readers who are familiar with the Web, HTTP, and Apache, but are not security experts. It is not intended to be a definitive guide to the SSL protocol, nor does it discuss specific techniques for managing certificates in an organization, or the important legal issues of patents and import and export restrictions. Rather, it is intended to provide a common background to mod_ssl users by pulling together various concepts, definitions, and examples as a starting point for further exploration.

The presented content is mainly derived, with permission by the author, from the article *Introducing SSL and Certificates using SSLeay* from Frederick J. Hirsch, of The Open Group Research Institute, which was published in *Web Security: A Matter of Trust*, World Wide Web Journal, Volume 2, Issue 3, Summer 1997. Please send any postive feedback to Frederick Hirsch (the original article author) and all negative feedback to Ralf S. Engelschall (the mod_ssl author).

Cryptographic Techniques

Understanding SSL requires an understanding of cryptographic algorithms, message digest functions (aka. one-way or hash functions), and digital signatures. These techniques are the subject of entire books (see for instance [AC96]) and provide the basis for privacy, integrity, and authentication.

Cryptographic Algorithms

Suppose Alice wants to send a message to her bank to transfer some money. Alice would like the message to be private, since it will include information such as her account number and transfer amount. One solution is to use a cryptographic algorithm, a technique that would transform her message into an encrypted form, unreadable except by those it is intended for. Once in this form, the message may only be interpreted through the use of a secret key. Without the key the message is useless: good cryptographic algorithms make it so difficult for intruders to decode the original text that it isn't worth their effort.

There are two categories of cryptographic algorithms: conventional and public key.

- *Conventional cryptography*, also known as symmetric cryptography, requires the sender and receiver to share a key: a secret piece of information that may be used to encrypt or decrypt a message. If this key is secret, then nobody other than the sender or receiver may read the message. If Alice and the bank know a secret key, then they may send each other private messages. The task of privately choosing a key before communicating, however, can be problematic.
- *Public key cryptography*, also known as asymmetric cryptography, solves the key exchange problem by defining an algorithm which uses two keys, each of which may be used to encrypt a message. If one key is used to encrypt a message then the other must be used to decrypt it. This makes it possible to receive secure messages by simply publishing one key (the public key) and keeping the other secret (the private key).

  Anyone may encrypt a message using the public key, but only the owner of the private key will be able to read it. In this way, Alice may send private messages to the owner of a key-pair (the bank), by encrypting it using their public key. Only the bank will be able to decrypt it.

Message Digests

Although Alice may encrypt her message to make it private, there is still a concern that someone might modify her original message or substitute it with a different one, in order to transfer the money to themselves, for instance. One way of guaranteeing the integrity of Alice's message is to create a concise summary of her message and send this to the bank as well. Upon receipt of the message, the bank creates its own summary and compares it with the one Alice sent. If they agree then the message was received intact.

A summary such as this is called a *message digest*, *one-way function* or *hash function*. Message digests are used to create short, fixed-length representations of longer, variable-length messages. Digest algorithms are designed to produce unique digests for different messages. Message digests are designed to make it too difficult to determine the message from the digest, and also impossible to find two different messages which create the same digest -- thus eliminating the possibility of substituting one message for another while maintaining the same digest.

Another challenge that Alice faces is finding a way to send the digest to the bank securely; when this is achieved, the integrity of the associated message is assured. One way to to this is to include the digest in a digital signature.

Digital Signatures

When Alice sends a message to the bank, the bank needs to ensure that the message is really from her, so an intruder does not request a transaction involving her account. A *digital signature*, created by Alice and included with the message, serves this purpose.

Digital signatures are created by encrypting a digest of the message, and other information (such as a sequence number) with the sender's private key. Though anyone may *decrypt* the signature using the public key, only the signer knows the private key. This means that only they may have signed it. Including the digest in the signature means the signature is only good for that message; it also ensures the integrity of the message since no one can change the digest and still sign it.

To guard against interception and reuse of the signature by an intruder at a later date, the signature contains a unique sequence number. This protects the bank from a fraudulent claim from Alice that she did not send the message -- only she could have signed it (non-repudiation).

Certificates

Although Alice could have sent a private message to the bank, signed it, and ensured the integrity of the message, she still needs to be sure that she is really communicating with the bank. This means that she needs to be sure that the public key she is using corresponds to the bank's private key. Similarly, the bank also needs to verify that the message signature really corresponds to Alice's signature.

If each party has a certificate which validates the other's identity, confirms the public key, and is signed by a trusted agency, then they both will be assured that they are communicating with whom they think they are. Such a trusted agency is called a *Certificate Authority*, and certificates are used for authentication.

Certificate Contents

A certificate associates a public key with the real identity of an individual, server, or other entity, known as the subject. As shown in Table 1, information about the subject includes identifying information (the distinguished name), and the public key. It also includes the identification and signature of the Certificate Authority that issued the certificate, and the period of time during which the certificate is valid. It may have additional information (or extensions) as well as administrative information for the Certificate Authority's use, such as a serial number.

Subject:	Distinguished Name, Public Key
Issuer:	Distinguished Name, Signature
Period of Validity:	Not Before Date, Not After Date
Administrative Information:	Version, Serial Number
Extended Information:	Basic Contraints, Netscape Flags, etc.

Table 1: Certificate Information

A distinguished name is used to provide an identity in a specific context -- for instance, an individual might have a personal certificate as well as one for their identity as an employee. Distinguished names are defined by the X.509 standard [X509], which defines the fields, field names, and abbreviations used to refer to the fields (see Table 2).

DN Field:	**Abbrev.:**	**Description:**	**Example:**
Common Name	CN	Name being certified	CN=Joe Average
Organization or Company	O	Name is associated with this organization	O=Snake Oil, Ltd.
Organizational Unit	OU	Name is associated with this organization unit, such as a department	OU=Research Institute
City/Locality	L	Name is located in this City	L=Snake City
State/Province	ST	Name is located in this State/Province	ST=Desert
Country	C	Name is located in this Country (ISO code)	C=XZ

Table 2: Distinguished Name Information

A Certificate Authority may define a policy specifying which distinguished field names are optional, and which are required. It may also place requirements upon the field contents, as may users of certificates. As an example, a Netscape browser requires that the Common Name for a certificate representing a server has a name which matches a wildcard pattern for the domain name of that server, such as `*.snakeoil.com`.

The binary format of a certificate is defined using the ASN.1 notation [X208] [PKCS]. This notation defines how to specify the contents, and encoding rules define how this information is translated into binary form. The binary encoding of the certificate is defined using Distinguished Encoding Rules (DER), which are based on the more general Basic Encoding Rules (BER). For those transmissions which cannot handle binary, the binary form may be translated into an ASCII form by using Base64 encoding [MIME]. This encoded version is called PEM encoded (the name comes from "Privacy Enhanced Mail"), when placed between begin and end delimiter lines as illustrated in Table 3.

```
-----BEGIN CERTIFICATE-----
MIIC7jCCAlegAwIBAgIBATANBgkqhkiG9w0BAQQFADCBqTELMAkGA1UEBhMCWFkx
FTATBgNVBAgTDFNuYWtlIERlc2VydDETMBEGA1UEBxMKU25ha2UgVG93bjEXMBUG
A1UEChMOU25ha2UgT2lsLCBMdGQxHjAcBgNVBAsTFUNlcnRpZmljYXRlIEF1dGhv
cml0eTEVMBMGA1UEAxMMU25ha2UgT2lsIENBMR4wHAYJKoZIhvcNAQkBFg9jYUBz
bmFrZW9pbC5kb20wHhcNOTgxMDIxMDg1ODM2WhcNOTkxMDIxMDg1ODM2WjCBpzEL
MAkGA1UEBhMCWFkxFTATBgNVBAgTDFNuYWtlIERlc2VydDETMBEGA1UEBxMKU25h
a2UgVG93bjEXMBUGA1UEChMOU25ha2UgT2lsLCBMdGQxFzAVBgNVBAsTDldlYnNl
cnZlciBUZWFtMRkwFwYDVQQDExB3d3cuc25ha2VvaWwuZG9tMR8wHQYJKoZIhvcN
AQkBFhB3d3dAc25ha2VvaWwuZG9tMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKB
gQDH9Ge/s2zcH+da+rPTx/DPRp3xGjHZ4GG6pCmvADIEtBtKBFAcZ64n+Dy7Np8b
vKR+yy5DGQiijsH1D/j8HlGE+q4TZ8OFk7BNBFazHxFbYI4OKMiCxdKzdif1yfaa
lWoANFlAzlSdbxeGVHoT0K+gT5w3UxwZKv2DLbCTzLZyPwIDAQABoyYwJDAPBgNV
HRMECDAGAQH/AgEAMBEGCWCGSAGG+EIBAQQEAwIAQDANBgkqhkiG9w0BAQQFAAOB
gQAZUIHAL4D09oE6Lv2k56Gp38OBDuILvwLg1v1KL8mQR+KFjghCrtpqaztZqcDt
2q2QoyulCgSzHbEGmi0EsdkPfg6mp0penssIFePYNI+/8u9HT4LuKMJX15hxBam7
dUHzICxBVC1lnHyYGjDuAMhe396lYAn8bCld1/L4NMGBCQ==
-----END CERTIFICATE-----
```

Table 3: Example of a PEM-encoded certificate (snakeoil.crt)

Certificate Authorities

By first verifying the information in a certificate request before granting the certificate, the Certificate Authority assures the identity of the private key owner of a key-pair. For instance, if Alice requests a personal certificate, the Certificate Authority must first make sure that Alice really is the person the certificate request claims.

Certificate Chains

A Certificate Authority may also issue a certificate for another Certificate Authority. When examining a certificate, Alice may need to examine the certificate of the issuer, for each parent Certificate Authority, until reaching one which she has confidence in. She may decide to trust only certificates with a limited chain of issuers, to reduce her risk of a "bad" certificate in the chain.

Creating a Root-Level CA

As noted earlier, each certificate requires an issuer to assert the validity of the identity of the certificate subject, up to the top-level Certificate Authority (CA). This presents a problem: Since this is who vouches for the certificate of the top-level authority, which has no issuer? In this unique case, the certificate is "self-signed", so the issuer of the certificate is the same as the subject. As a result, one must exercise extra care in trusting a self-signed certificate. The wide publication of a public key by the root authority reduces the risk in trusting this key -- it would be obvious if someone else publicized a key claiming to be the authority. Browsers are preconfigured to trust well-known certificate authorities.

A number of companies, such as Thawte and VeriSign have established themselves as Certificate Authorities. These companies provide the following services:

- Verifying certificate requests
- Processing certificate requests
- Issuing and managing certificates

It is also possible to create your own Certificate Authority. Although risky in the Internet environment, it may be useful within an Intranet where the organization can easily verify the identities of individuals and servers.

Certificate Management

Establishing a Certificate Authority is a responsibility which requires a solid administrative, technical, and management framework. Certificate Authorities not only issue certificates, they also manage them -- that is, they determine how long certificates are valid, they renew them, and they keep lists of certificates that have already been issued but are no longer valid (Certificate Revocation Lists, or CRLs). Say Alice is entitled to a certificate as an employee of a company. Say too, that the certificate needs to be revoked when Alice leaves the company. Since certificates are objects that get passed around, it is impossible to tell from the certificate alone that it has been revoked. When examining certificates for validity, therefore, it is necessary to contact the issuing Certificate Authority to check CRLs -- this is not usually an automated part of the process.

Note:

If you use a Certificate Authority that is not configured into browsers by default, it is necessary to load the Certificate Authority certificate into the browser, enabling the browser to validate server certificates signed by that Certificate Authority. Doing so may be dangerous, since once loaded, the browser will accept all certificates signed by that Certificate Authority.

Secure Sockets Layer (SSL)

The Secure Sockets Layer protocol is a protocol layer which may be placed between a reliable connection-oriented network layer protocol (e.g. TCP/IP) and the application protocol layer (e.g. HTTP). SSL provides for secure communication between client and server by allowing mutual authentication, the use of digital signatures for integrity, and encryption for privacy.

The protocol is designed to support a range of choices for specific algorithms used for cryptography, digests, and signatures. This allows algorithm selection for specific servers to be made based on legal, export or other concerns, and also enables the protocol to take advantage of new algorithms. Choices are negotiated between client and server at the start of establishing a protocol session.

Version:	Source:	Description:	Browser Support:
SSL v2.0	Vendor Standard (from Netscape Corp.) [SSL2]	First SSL protocol for which implementations exists	- NS Navigator 1.x/2.x - MS IE 3.x - Lynx/2.8+OpenSSL
SSL v3.0	Expired Internet Draft (from Netscape Corp.) [SSL3]	Revisions to prevent specific security attacks, add non-RSA ciphers, and support for certificate chains	- NS Navigator 2.x/3.x/4.x - MS IE 3.x/4.x - Lynx/2.8+OpenSSL
TLS v1.0	Proposed Internet Standard (from IETF) [TLS1]	Revision of SSL 3.0 to update the MAC layer to HMAC, add block padding for block ciphers, message order standardization and more alert messages.	- Lynx/2.8+OpenSSL

Table 4: Versions of the SSL protocol

There are a number of versions of the SSL protocol, as shown in Table 4. As noted there, one of the benefits in SSL 3.0 is that it adds support of certificate chain loading. This feature allows a server to pass a server certificate along with issuer certificates to the browser. Chain loading also permits the browser to validate the server certificate, even if Certificate Authority certificates are not installed for the intermediate issuers, since they are included in the certificate chain. SSL 3.0 is the basis for the Transport Layer Security [TLS] protocol standard, currently in development by the Internet Engineering Task Force (IETF).

Session Establishment

The SSL session is established by following a *handshake sequence* between client and server, as shown in Figure 1. This sequence may vary, depending on whether the server is configured to provide a server certificate or request a client certificate. Though cases exist where additional handshake steps are required for management of cipher information, this article summarizes one common scenario: see the SSL specification for the full range of possibilities.

Note

Once an SSL session has been established it may be reused, thus avoiding the performance penalty of repeating the many steps needed to start a session. For this the server assigns each SSL session a unique session identifier which is cached in the server and which the client can use on forthcoming connections to reduce the handshake (until the session identifer expires in the cache of the server).

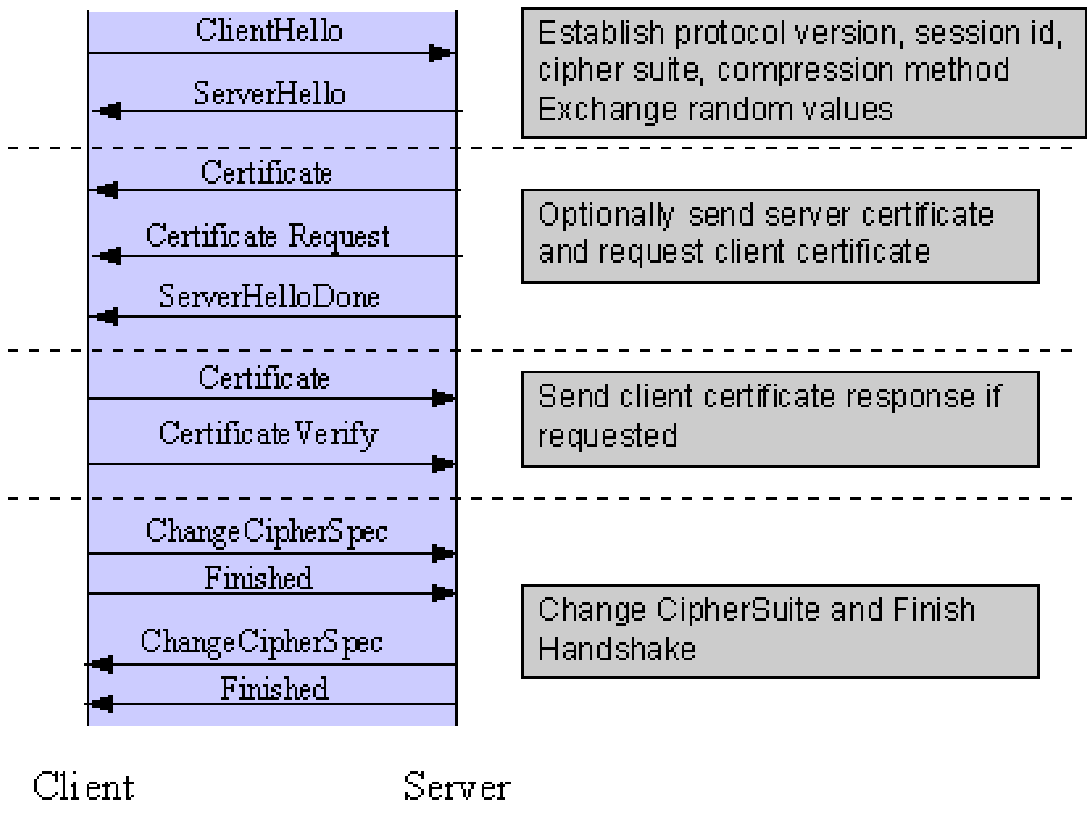

ClientHello
ServerHello
Establish protocol version, session id, cipher suite, compression method Exchange random values
Certificate
Certificate Request
ServerHelloDone
Optionally send server certificate and request client certificate
Certificate
Certificate Verify
Send client certificate response if requested
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished
Change CipherSuite and Finish Handshake
Client
Server

Figure 1: Simplified SSL Handshake Sequence

The elements of the handshake sequence, as used by the client and server, are listed below:

1. Negotiate the Cipher Suite to be used during data transfer
2. Establish and share a session key between client and server
3. Optionally authenticate the server to the client
4. Optionally authenticate the client to the server

The first step, Cipher Suite Negotiation, allows the client and server to choose a Cipher Suite supportable by both of them. The SSL3.0 protocol specification defines 31 Cipher Suites. A Cipher Suite is defined by the following components:

- Key Exchange Method
- Cipher for Data Transfer
- Message Digest for creating the Message Authentication Code (MAC)

These three elements are described in the sections that follow.

Key Exchange Method

The key exchange method defines how the shared secret symmetric cryptography key used for application data transfer will be agreed upon by client and server. SSL 2.0 uses RSA key exchange only, while SSL 3.0 supports a choice of key exchange algorithms including the RSA key exchange when certificates are used, and Diffie-Hellman key exchange for exchanging keys without certificates and without prior communication between client and server.

One variable in the choice of key exchange methods is digital signatures -- whether or not to use them, and if so, what kind of signatures to use. Signing with a private key provides assurance against a man-in-the-middle-attack during the information exchange used in generating the shared key [AC96, p516].

Cipher for Data Transfer

SSL uses the conventional cryptography algorithm (symmetric cryptography) described earlier for encrypting messages in a session. There are nine choices, including the choice to perform no encryption:

- No encryption
- Stream Ciphers
  - RC4 with 40-bit keys
  - RC4 with 128-bit keys
- CBC Block Ciphers
  - RC2 with 40 bit key
  - DES with 40 bit key
  - DES with 56 bit key
  - Triple-DES with 168 bit key
  - Idea (128 bit key)
  - Fortezza (96 bit key)

Here "CBC" refers to Cipher Block Chaining, which means that a portion of the previously encrypted cipher text is used in the encryption of the current block. "DES" refers to the Data Encryption Standard [AC96, ch12], which has a number of variants (including DES40 and 3DES_EDE). "Idea" is one of the best and cryptographically strongest available algorithms, and "RC2" is a proprietary algorithm from RSA DSI [AC96, ch13].

Digest Function

The choice of digest function determines how a digest is created from a record unit. SSL supports the following:

- No digest (Null choice)
- MD5, a 128-bit hash
- Secure Hash Algorithm (SHA-1), a 160-bit hash

The message digest is used to create a Message Authentication Code (MAC) which is encrypted with the message to provide integrity and to prevent against replay attacks.

Handshake Sequence Protocol

The handshake sequence uses three protocols:

- The *SSL Handshake Protocol* for performing the client and server SSL session establishment.
- The *SSL Change Cipher Spec Protocol* for actually establishing agreement on the Cipher Suite for the session.
- The *SSL Alert Protocol* for conveying SSL error messages between client and server.

These protocols, as well as application protocol data, are encapsulated in the *SSL Record Protocol*, as shown in Figure 2. An encapsulated protocol is transferred as data by the lower layer protocol, which does not examine the data. The encapsulated protocol has no knowledge of the underlying protocol.

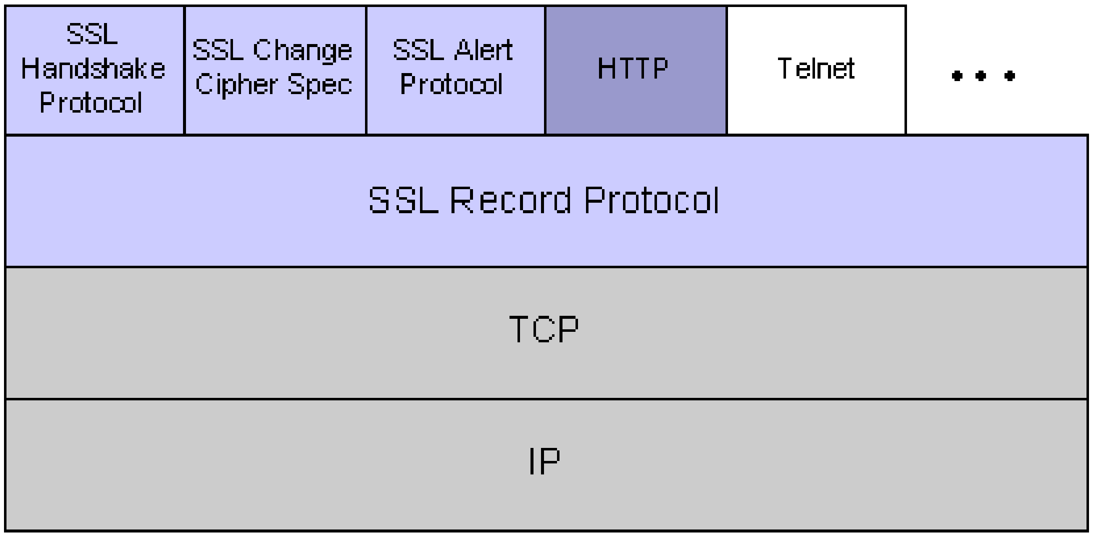


Figure 2: SSL Protocol Stack

The encapsulation of SSL control protocols by the record protocol means that if an active session is renegotiated the control protocols will be transmitted securely. If there were no session before, then the Null cipher suite is used, which means there is no encryption and messages have no integrity digests until the session has been established.

Data Transfer

The SSL Record Protocol, shown in Figure 3, is used to transfer application and SSL Control data between the client and server, possibly fragmenting this data into smaller units, or combining multiple higher level protocol data messages into single units. It may compress, attach digest signatures, and encrypt these units before transmitting them using the underlying reliable transport protocol (Note: currently all major SSL implementations lack support for compression).

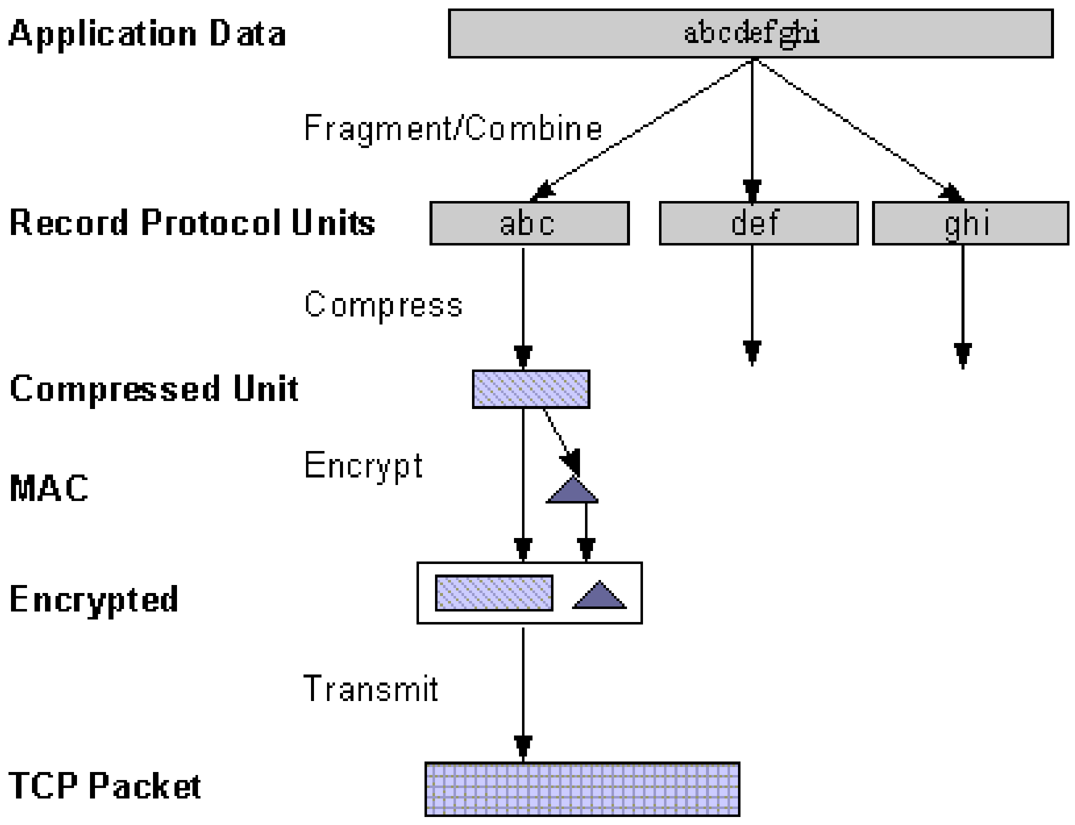


Figure 3: SSL Record Protocol

Securing HTTP Communication

One common use of SSL is to secure Web HTTP communication between a browser and a webserver. This case does not preclude the use of non-secured HTTP. The secure version is mainly plain HTTP over SSL (named HTTPS), but with one major difference: it uses the URL scheme `https` rather than `http` and a different server port (by default 443). This mainly is what mod_ssl provides to you for the Apache webserver...

References

- [AC96] Bruce Schneier, *Applied Cryptography*, 2nd Edition, Wiley, 1996. See http://www.counterpane.com/ for various other materials by Bruce Schneier.
- [X208] ITU-T Recommendation X.208, *Specification of Abstract Syntax Notation One (ASN.1)*, 1988. See for instance

ftp://ftp.neda.com/pub/itu/x.series/x208.ps.

- [X509] ITU-T Recommendation X.509, *The Directory - Authentication Framework*, 1988. See for instance ftp://ftp.bull.com/pub/OSIdirectory/ITUnov96/X.509/97x509final.doc.
- [PKCS] Kaliski, Burton S., Jr., *An Overview of the PKCS Standards*, An RSA Laboratories Technical Note, revised November 1, 1993. See http://www.rsa.com/rsalabs/pubs/PKCS/.
- [MIME] N. Freed, N. Borenstein, *Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies*, RFC2045. See for instance ftp://ftp.isi.edu/in-notes/rfc2045.txt.
- [SSL2] Kipp E.B. Hickman, *The SSL Protocol*, 1995. See http://www.netscape.com/eng/security/SSL_2.html.
- [SSL3] Alan O. Freier, Philip Karlton, Paul C. Kocher, *The SSL Protocol Version 3.0*, 1996. See http://www.netscape.com/eng/ssl3/draft302.txt.
- [TLS1] Tim Dierks, Christopher Allen, *The TLS Protocol Version 1.0*, 1997. See ftp://ftp.ietf.org/internet-drafts/draft-ietf-tls-protocol-06.txt.

Apache HTTP Server Version 2.0

SSL/TLS Strong Encryption: Compatibility

All PCs are compatible. But some of them are more compatible than others.
Unknown

Here we talk about backward compatibility to other SSL solutions. As you perhaps know, mod_ssl is not the only existing SSL solution for Apache. Actually there are four additional major products available on the market: Ben Laurie's freely available Apache-SSL (from where mod_ssl were originally derived in 1998), RedHat's commercial Secure Web Server (which is based on mod_ssl), Covalent's commercial Raven SSL Module (also based on mod_ssl) and finally C2Net's commercial product Stronghold (based on a different evolution branch named Sioux up to Stronghold 2.x and based on mod_ssl since Stronghold 3.x).

The idea in mod_ssl is mainly the following: because mod_ssl provides mostly a superset of the functionality of all other solutions we can easily provide backward compatibility for most of the cases. Actually there are three compatibility areas we currently address: configuration directives, environment variables and custom log functions.

Configuration Directives

For backward compatibility to the configuration directives of other SSL solutions we do an on-the-fly mapping: directives which have a direct counterpart in mod_ssl are mapped silently while other directives lead to a warning message in the logfiles. The currently implemented directive mapping is listed in Table 1. Currently full backward compatibilty is provided only for Apache-SSL 1.x and mod_ssl 2.0.x. Compatibility to Sioux 1.x and Stronghold 2.x is only partial because of special functionality in these interfaces which mod_ssl (still) doesn't provide.

Old Directive	**mod_ssl Directive**	**Comment**
Apache-SSL 1.x & mod_ssl 2.0.x compatibility:		
`SSLEnable`	`SSLEngine on`	compactified
`SSLDisable`	`SSLEngine off`	compactified
`SSLLogFile` *file*	`SSLLog` *file*	compactified
`SSLRequiredCiphers` *spec*	`SSLCipherSuite` *spec*	renamed
`SSLRequireCipher` *c1* ...	`SSLRequire %{SSL_CIPHER} in {"`*c1*`", ...}`	generalized
`SSLBanCipher` *c1* ...	`SSLRequire not (%{SSL_CIPHER} in {"`*c1*`", ...})`	generalized
`SSLFakeBasicAuth`	`SSLOptions +FakeBasicAuth`	merged
`SSLCacheServerPath` *dir*	-	functionality removed
`SSLCacheServerPort` *integer*	-	functionality removed
Apache-SSL 1.x compatibility:		
`SSLExportClientCertificates`	`SSLOptions +ExportCertData`	merged
`SSLCacheServerRunDir` *dir*	-	functionality not supported
Sioux 1.x compatibility:		
`SSL_CertFile` *file*	`SSLCertificateFile` *file*	renamed
`SSL_KeyFile` *file*	`SSLCertificateKeyFile` *file*	renamed
`SSL_CipherSuite` *arg*	`SSLCipherSuite` *arg*	renamed
`SSL_X509VerifyDir` *arg*	`SSLCACertificatePath` *arg*	renamed
`SSL_Log` *file*	`SSLLogFile` *file*	renamed
`SSL_Connect` *flag*	`SSLEngine` *flag*	renamed
`SSL_ClientAuth` *arg*	`SSLVerifyClient` *arg*	renamed

`SSL_X509VerifyDepth` *arg*	`SSLVerifyDepth` *arg*	renamed
`SSL_FetchKeyPhraseFrom` *arg*	-	not directly mappable; use SSLPassPhraseDialog
`SSL_SessionDir` *dir*	-	not directly mappable; use SSLSessionCache
`SSL_Require` *expr*	-	not directly mappable; use SSLRequire
`SSL_CertFileType` *arg*	-	functionality not supported
`SSL_KeyFileType` *arg*	-	functionality not supported
`SSL_X509VerifyPolicy` *arg*	-	functionality not supported
`SSL_LogX509Attributes` *arg*	-	functionality not supported
Stronghold 2.x compatibility:		
`StrongholdAccelerator` *dir*	-	functionality not supported
`StrongholdKey` *dir*	-	functionality not supported
`StrongholdLicenseFile` *dir*	-	functionality not supported
`SSLFlag` *flag*	`SSLEngine` *flag*	renamed
`SSLSessionLockFile` *file*	`SSLMutex` *file*	renamed
`SSLCipherList` *spec*	`SSLCipherSuite` *spec*	renamed
`RequireSSL`	`SSLRequireSSL`	renamed
`SSLErrorFile` *file*	-	functionality not supported
`SSLRoot` *dir*	-	functionality not supported
`SSL_CertificateLogDir` *dir*	-	functionality not supported
`AuthCertDir` *dir*	-	functionality not supported
`SSL_Group` *name*	-	functionality not supported
`SSLProxyMachineCertPath` *dir*	-	functionality not supported
`SSLProxyMachineCertFile` *file*	-	functionality not supported
`SSLProxyCACertificatePath` *dir*	-	functionality not supported
`SSLProxyCACertificateFile` *file*	-	functionality not supported
`SSLProxyVerifyDepth` *number*	-	functionality not supported
`SSLProxyCipherList` *spec*	-	functionality not supported

Table 1: Configuration Directive Mapping

Environment Variables

When you use ``` ``SSLOptions +CompatEnvVars" ``` additional environment variables are generated. They all correspond to existing official mod_ssl variables. The currently implemented variable derivation is listed in Table 2.

Old Variable	**mod_ssl Variable**	**Comment**
`SSL_PROTOCOL_VERSION`	`SSL_PROTOCOL`	renamed
`SSLEAY_VERSION`	`SSL_VERSION_LIBRARY`	renamed
`HTTPS_SECRETKEYSIZE`	`SSL_CIPHER_USEKEYSIZE`	renamed
`HTTPS_KEYSIZE`	`SSL_CIPHER_ALGKEYSIZE`	renamed
`HTTPS_CIPHER`	`SSL_CIPHER`	renamed
`HTTPS_EXPORT`	`SSL_CIPHER_EXPORT`	renamed
`SSL_SERVER_KEY_SIZE`	`SSL_CIPHER_ALGKEYSIZE`	renamed
`SSL_SERVER_CERTIFICATE`	`SSL_SERVER_CERT`	renamed
`SSL_SERVER_CERT_START`	`SSL_SERVER_V_START`	renamed
`SSL_SERVER_CERT_END`	`SSL_SERVER_V_END`	renamed
`SSL_SERVER_CERT_SERIAL`	`SSL_SERVER_M_SERIAL`	renamed
`SSL_SERVER_SIGNATURE_ALGORITHM`	`SSL_SERVER_A_SIG`	renamed
`SSL_SERVER_DN`	`SSL_SERVER_S_DN`	renamed
`SSL_SERVER_CN`	`SSL_SERVER_S_DN_CN`	renamed
`SSL_SERVER_EMAIL`	`SSL_SERVER_S_DN_Email`	renamed
`SSL_SERVER_O`	`SSL_SERVER_S_DN_O`	renamed

`SSL_SERVER_OU`	`SSL_SERVER_S_DN_OU`	renamed
`SSL_SERVER_C`	`SSL_SERVER_S_DN_C`	renamed
`SSL_SERVER_SP`	`SSL_SERVER_S_DN_SP`	renamed
`SSL_SERVER_L`	`SSL_SERVER_S_DN_L`	renamed
`SSL_SERVER_IDN`	`SSL_SERVER_I_DN`	renamed
`SSL_SERVER_ICN`	`SSL_SERVER_I_DN_CN`	renamed
`SSL_SERVER_IEMAIL`	`SSL_SERVER_I_DN_Email`	renamed
`SSL_SERVER_IO`	`SSL_SERVER_I_DN_O`	renamed
`SSL_SERVER_IOU`	`SSL_SERVER_I_DN_OU`	renamed
`SSL_SERVER_IC`	`SSL_SERVER_I_DN_C`	renamed
`SSL_SERVER_ISP`	`SSL_SERVER_I_DN_SP`	renamed
`SSL_SERVER_IL`	`SSL_SERVER_I_DN_L`	renamed
`SSL_CLIENT_CERTIFICATE`	`SSL_CLIENT_CERT`	renamed
`SSL_CLIENT_CERT_START`	`SSL_CLIENT_V_START`	renamed
`SSL_CLIENT_CERT_END`	`SSL_CLIENT_V_END`	renamed
`SSL_CLIENT_CERT_SERIAL`	`SSL_CLIENT_M_SERIAL`	renamed
`SSL_CLIENT_SIGNATURE_ALGORITHM`	`SSL_CLIENT_A_SIG`	renamed
`SSL_CLIENT_DN`	`SSL_CLIENT_S_DN`	renamed
`SSL_CLIENT_CN`	`SSL_CLIENT_S_DN_CN`	renamed
`SSL_CLIENT_EMAIL`	`SSL_CLIENT_S_DN_Email`	renamed
`SSL_CLIENT_O`	`SSL_CLIENT_S_DN_O`	renamed
`SSL_CLIENT_OU`	`SSL_CLIENT_S_DN_OU`	renamed
`SSL_CLIENT_C`	`SSL_CLIENT_S_DN_C`	renamed
`SSL_CLIENT_SP`	`SSL_CLIENT_S_DN_SP`	renamed
`SSL_CLIENT_L`	`SSL_CLIENT_S_DN_L`	renamed
`SSL_CLIENT_IDN`	`SSL_CLIENT_I_DN`	renamed
`SSL_CLIENT_ICN`	`SSL_CLIENT_I_DN_CN`	renamed
`SSL_CLIENT_IEMAIL`	`SSL_CLIENT_I_DN_Email`	renamed
`SSL_CLIENT_IO`	`SSL_CLIENT_I_DN_O`	renamed
`SSL_CLIENT_IOU`	`SSL_CLIENT_I_DN_OU`	renamed
`SSL_CLIENT_IC`	`SSL_CLIENT_I_DN_C`	renamed
`SSL_CLIENT_ISP`	`SSL_CLIENT_I_DN_SP`	renamed
`SSL_CLIENT_IL`	`SSL_CLIENT_I_DN_L`	renamed
`SSL_EXPORT`	`SSL_CIPHER_EXPORT`	renamed
`SSL_KEYSIZE`	`SSL_CIPHER_ALGKEYSIZE`	renamed
`SSL_SECKEYSIZE`	`SSL_CIPHER_USEKEYSIZE`	renamed
`SSL_SSLEAY_VERSION`	`SSL_VERSION_LIBRARY`	renamed
`SSL_STRONG_CRYPTO`	-	Not supported by mod_ssl
`SSL_SERVER_KEY_EXP`	-	Not supported by mod_ssl
`SSL_SERVER_KEY_ALGORITHM`	-	Not supported by mod_ssl
`SSL_SERVER_KEY_SIZE`	-	Not supported by mod_ssl
`SSL_SERVER_SESSIONDIR`	-	Not supported by mod_ssl
`SSL_SERVER_CERTIFICATELOGDIR`	-	Not supported by mod_ssl
`SSL_SERVER_CERTFILE`	-	Not supported by mod_ssl
`SSL_SERVER_KEYFILE`	-	Not supported by mod_ssl
`SSL_SERVER_KEYFILETYPE`	-	Not supported by mod_ssl
`SSL_CLIENT_KEY_EXP`	-	Not supported by mod_ssl
`SSL_CLIENT_KEY_ALGORITHM`	-	Not supported by mod_ssl
`SSL_CLIENT_KEY_SIZE`	-	Not supported by mod_ssl

Table 2: Environment Variable Derivation

Custom Log Functions

When mod_ssl is built into Apache or at least loaded (under DSO situation) additional functions exist for the Custom Log Format of mod_log_config as documented in the Reference Chapter. Beside the ``%{*varname*}x" eXtension format function which can be used to expand any variables provided by any module, an additional Cryptography ``%{*name*}c" cryptography format function exists for backward compatibility. The currently implemented function calls are listed in Table 3.

Function Call	**Description**
`%...{version}c`	SSL protocol version
`%...{cipher}c`	SSL cipher
`%...{subjectdn}c`	Client Certificate Subject Distinguished Name
`%...{issuerdn}c`	Client Certificate Issuer Distinguished Name
`%...{errcode}c`	Certificate Verification Error (numerical)
`%...{errstr}c`	Certificate Verification Error (string)

Table 3: Custom Log Cryptography Function

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

SSL/TLS Strong Encryption: How-To

``The solution of this problem is trivial and is left as an exercise for the reader.''
Standard textbook cookie

How to solve particular security constraints for an SSL-aware webserver is not always obvious because of the coherences between SSL, HTTP and Apache's way of processing requests. This chapter gives instructions on how to solve such typical situations. Treat is as a first step to find out the final solution, but always try to understand the stuff before you use it. Nothing is worse than using a security solution without knowing its restrictions and coherences.

Cipher Suites and Enforced Strong Security

- **How can I create a real SSLv2-only server?** [L]

  The following creates an SSL server which speaks only the SSLv2 protocol and its ciphers.

  httpd.conf

  ```
  SSLProtocol -all +SSLv2
  SSLCipherSuite SSLv2:+HIGH:+MEDIUM:+LOW:+EXP
  ```

- **How can I create an SSL server which accepts strong encryption only?** [L]

  The following enables only the seven strongest ciphers:

  httpd.conf

  ```
  SSLProtocol all
  SSLCipherSuite HIGH:MEDIUM
  ```

- **How can I create an SSL server which accepts strong encryption only, but allows export browsers to upgrade to stronger encryption?** [L]

This facility is called Server Gated Cryptography (SGC) and details you can find in the `README.GlobalID` document in the mod_ssl distribution. In short: The server has a Global ID server certificate, signed by a special CA certificate from Verisign which enables strong encryption in export browsers. This works as following: The browser connects with an export cipher, the server sends its Global ID certificate, the browser verifies it and subsequently upgrades the cipher suite before any HTTP communication takes place. The question now is: How can we allow this upgrade, but enforce strong encryption. Or in other words: Browser either have to initially connect with strong encryption or have to upgrade to strong encryption, but are not allowed to keep the export ciphers. The following does the trick:

httpd.conf

```
#   allow all ciphers for the inital handshake,
#   so export browsers can upgrade via SGC facility
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
<Directory /usr/local/apache/htdocs>
#   but finally deny all browsers which haven't upgraded
SSLRequire %{SSL_CIPHER_USEKEYSIZE} >= 128
</Directory>
```

- **How can I create an SSL server which accepts all types of ciphers in general, but requires a strong ciphers for access to a particular URL?** [L]

Obviously you cannot just use a server-wide `SSLCipherSuite` which restricts the ciphers to the strong variants. But mod_ssl allows you to reconfigure the cipher suite in per-directory context and automatically forces a renegotiation of the SSL parameters to meet the new configuration. So, the solution is:

httpd.conf

```
#   be liberal in general
SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP:+eNULL
<Location /strong/area>
#   but https://hostname/strong/area/ and below requires strong ciphers
SSLCipherSuite HIGH:MEDIUM
</Location>
```

Client Authentication and Access Control

- **How can I authenticate clients based on certificates when I know all my clients?** [L]

When you know your user community (i.e. a closed user group situation), as it's the case for instance in an Intranet, you can use plain certificate authentication. All you have to do is to create client certificates signed by your own CA certificate `ca.crt` and then verifiy the clients against this certificate.

httpd.conf

```
#   require a client certificate which has to be directly
#   signed by our CA certificate in ca.crt
SSLVerifyClient require
SSLVerifyDepth 1
SSLCACertificateFile conf/ssl.crt/ca.crt
```

- **How can I authenticate my clients for a particular URL based on certificates but still allow arbitrary clients to access the remaining parts of the server?** [L]

For this we again use the per-directory reconfiguration feature of mod_ssl:

httpd.conf

```
SSLVerifyClient none
SSLCACertificateFile conf/ssl.crt/ca.crt
<Location /secure/area>
SSLVerifyClient require
SSLVerifyDepth 1
</Location>
```

- **How can I authenticate only particular clients for a some URLs based on certificates but still allow arbitrary clients to access the remaining parts of the server?** [L]

  The key is to check for various ingredients of the client certficate. Usually this means to check the whole or part of the Distinguished Name (DN) of the Subject. For this two methods exists: The `mod_auth` based variant and the `SSLRequire` variant. The first method is good when the clients are of totally different type, i.e. when their DNs have no common fields (usually the organisation, etc.). In this case you've to establish a password database containing *all* clients. The second method is better when your clients are all part of a common hierarchy which is encoded into the DN. Then you can match them more easily.

  The first method:

  httpd.conf

```
SSLVerifyClient      none
<Directory /usr/local/apache/htdocs/secure/area>
SSLVerifyClient      require
SSLVerifyDepth       5
SSLCACertificateFile conf/ssl.crt/ca.crt
SSLCACertificatePath conf/ssl.crt
SSLOptions           +FakeBasicAuth
SSLRequireSSL
AuthName             "Snake Oil Authentication"
AuthType             Basic
AuthUserFile         /usr/local/apache/conf/httpd.passwd
require              valid-user
</Directory>
```

  httpd.passwd

```
/C=DE/L=Munich/O=Snake Oil, Ltd./OU=Staff/CN=Foo:xxj31ZMTZzkVA
/C=US/L=S.F./O=Snake Oil, Ltd./OU=CA/CN=Bar:xxj31ZMTZzkVA
/C=US/L=L.A./O=Snake Oil, Ltd./OU=Dev/CN=Quux:xxj31ZMTZzkVA
```

  The second method:

  httpd.conf

```
SSLVerifyClient      none
<Directory /usr/local/apache/htdocs/secure/area>
SSLVerifyClient      require
SSLVerifyDepth       5
SSLCACertificateFile conf/ssl.crt/ca.crt
SSLCACertificatePath conf/ssl.crt
SSLOptions           +FakeBasicAuth
SSLRequireSSL
SSLRequire           %{SSL_CLIENT_S_DN_O}  eq "Snake Oil, Ltd." and \
                     %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"}
</Directory>
```

- **How can I require HTTPS with strong ciphers and either basic authentication or client certificates for access to a subarea on the**

Intranet website for clients coming from the Internet but still allow plain HTTP access for clients on the Intranet? [L]

Let us assume the Intranet can be distinguished through the IP network 192.160.1.0/24 and the subarea on the Intranet website has the URL `/subarea`. Then configure the following outside your HTTPS virtual host (so it applies to both HTTPS and HTTP):

httpd.conf

```
SSLCACertificateFile conf/ssl.crt/company-ca.crt

<Directory /usr/local/apache/htdocs>
#   Outside the subarea only Intranet access is granted
Order                  deny,allow
Deny                   from all
Allow                  from 192.168.1.0/24
</Directory>

<Directory /usr/local/apache/htdocs/subarea>
#   Inside the subarea any Intranet access is allowed
#   but from the Internet only HTTPS + Strong-Cipher + Password
#   or the alternative HTTPS + Strong-Cipher + Client-Certificate

#   If HTTPS is used, make sure a strong cipher is used.
#   Additionally allow client certs as alternative to basic auth.
SSLVerifyClient        optional
SSLVerifyDepth         1
SSLOptions             +FakeBasicAuth +StrictRequire
SSLRequire             %{SSL_CIPHER_USEKEYSIZE} >= 128

#   Force clients from the Internet to use HTTPS
RewriteEngine          on
RewriteCond            %{REMOTE_ADDR} !^192\.168\.1\.[0-9]+$
RewriteCond            %{HTTPS} !=on
RewriteRule            .* - [F]

#   Allow Network Access and/or Basic Auth
Satisfy                any

#   Network Access Control
Order                  deny,allow
Deny                   from all
Allow                  192.168.1.0/24

#   HTTP Basic Authentication
AuthType               basic
AuthName               "Protected Intranet Area"
AuthUserFile           conf/protected.passwd
Require                valid-user
</Directory>
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

SSL/TLS Strong Encryption: FAQ

``The wise man doesn't give the right answers, he poses the right questions.''
Claude Levi-Strauss

This chapter is a collection of frequently asked questions (FAQ) and corresponding answers following the popular USENET tradition. Most of these questions occured on the Newsgroup `comp.infosystems.www.servers.unix` or the mod_ssl Support Mailing List `modssl-users@modssl.org`. They are collected at this place to avoid answering the same questions over and over.

Please read this chapter at least once when installing mod_ssl or at least search for your problem here before submitting a problem report to the author.

About the module

- **What is the history of mod_ssl?** [**L**]

  The mod_ssl v1 package was initially created in April 1998 by Ralf S. Engelschall via porting Ben Laurie's Apache-SSL 1.17 source patches for Apache 1.2.6 to Apache 1.3b6. Because of conflicts with Ben Laurie's development cycle it then was re-assembled from scratch for Apache 1.3.0 by merging the old mod_ssl 1.x with the newer Apache-SSL 1.18. From this point on mod_ssl lived its own life as mod_ssl v2. The first publically released version was mod_ssl 2.0.0 from August 10th, 1998. As of this writing (August 1999) the current mod_ssl version is 2.4.0.

  After one year of very active development with over 1000 working hours and over 40 releases mod_ssl reached its current state. The result is an already very clean source base implementing a very rich functionality. The code size increased by a factor of 4 to currently a total of over 10.000 lines of ANSI C consisting of approx. 70% code and 30% code documentation. From the original Apache-SSL code currently approx. 5% is remaining only.

  After the US export restrictions for cryptographic software were opened, mod_ssl was integrated into the code base of Apache V2 in 2001.
- **What are the functional differences between mod_ssl and Apache-SSL, from which it is originally derived?** [**L**]

This neither can be answered in short (there were too many code changes) nor can be answered at all by the author (there would immediately be flame wars with no reasonable results at the end). But as you easily can guess from the 5% of remaining Apache-SSL code, a lot of differences exists, although user-visible backward compatibility exists for most things.

When you really want a detailed comparison you have to read the entries in the large `CHANGES` file that is in the mod_ssl distribution. Usually this is much too hard-core. So I recommend you to either believe in the opinion and recommendations of other users (the simplest approach) or do a comparison yourself (the most reasonable approach). For the latter, grab distributions of mod_ssl (from http://www.modssl.org) and Apache-SSL (from http://www.apache-ssl.org), install both packages, read their documentation and try them out yourself. Then choose the one which pleases you most.

A few final hints to help direct your comparison: quality of documentation ("can you easily find answers and are they sufficient?"), quality of source code ("is the source code reviewable so you can make sure there aren't any trapdoors or inherent security risks because of bad programming style?"), easy and clean installation ("can the SSL functionality easily added to an Apache source tree without manual editing or patching?"), clean integration into Apache ("is the SSL functionality encapsulated and cleanly separated from the remaining Apache functionality?"), support for Dynamic Shared Object (DSO) facility ("can the SSL functionality built as a separate DSO for maximum flexibility?"), Win32 port ("is the SSL functionality available also under the Win32 platform?"), amount and quality of functionality ("is the provided SSL functionality and control possibilities sufficient for your situation?"), quality of problem tracing ("is it possible for you to easily trace down the problems via logfiles, etc?"), etc. pp.

- **What are the major differences between mod_ssl and the commercial alternatives like Raven or Stronghold?** [L]

  In the past (until September 20th, 2000) the major difference was the RSA license which one received (very cheaply in contrast to a direct licensing from RSA DSI) with the commercial Apache SSL products. On the other hand, one needed this license only in the US, of course. So for non-US citizens this point was useless. But now even for US citizens the situations changed because the RSA patent expired on September 20th, 2000 and RSA DSI also placed the RSA algorithm explicitly into the public domain.

  Second, there is the point that one has guaranteed support from the commercial vendors. On the other hand, if you monitored the Open Source quality of mod_ssl and the support activities found on `modssl-users@modssl.org`, you could ask yourself whether you are really convinced that you can get better support from a commercial vendor.

  Third, people often think they would receive perhaps at least a better technical SSL solution than mod_ssl from the commercial vendors. But this is not really true, because all commercial alternatives (Raven 1.4.x, Stronghold 3.x, RedHat SWS 2.x, etc.) *are* actually based on mod_ssl and OpenSSL. The reason for this common misunderstanding is mainly because some vendors make no attempt to make it reasonably clear that their product is actually mod_ssl based. So, do not think, just because the commercial alternatives are usually more expensive, that you are also receiving an alternative *technical* SSL solution. This is usually not the case. Actually the vendor versions of Apache, mod_ssl and OpenSSL often stay behind the latest free versions and perhaps this way still do not include important bug and security fixes. On the other hand, it sometimes occurs that a vendor version includes useful changes which are not available through the official freely available packages. But most vendors play fair and contribute back those changes to the free software world, of course.

  So, in short: There are lots of commercial versions of the popular Apache+mod_ssl+OpenSSL server combination available. Every user should decide carefully whether they really need to buy a commercial version or whether it would not be sufficient to directly use the free and official versions of the Apache, mod_ssl and OpenSSL packages.

- **How do I know which mod_ssl version is for which Apache version?** [L]

  That's trivial: mod_ssl uses version strings of the syntax <*mod_ssl-version*>-<*apache-version*>, for instance `2.4.0-1.3.9`. This directly indicates that it's mod_ssl version 2.4.0 for Apache version 1.3.9. And this also means you *only* can apply this mod_ssl version to exactly this Apache version (unless you use the `--force` option to mod_ssl's `configure` command ;-).

- **Is mod_ssl Year 2000 compliant?** [L]

  Yes, mod_ssl is Year 2000 compliant.

  Because first mod_ssl internally never stores years as two digits. Instead it always uses the ANSI C & POSIX numerical data type `time_t` type, which on almost all Unix platforms at the moment is a `signed long` (usually 32-bits) representing seconds since epoch of January 1st, 1970, 00:00 UTC. This signed value overflows in early January 2038 and not in the year 2000. Second, date and time presentations (for instance the variable ``` ``%{TIME_YEAR} ```") are done with full year value instead of abbreviating to two digits.

  Additionally according to a Year 2000 statement from the Apache Group, the Apache webserver is Year 2000 compliant, too. But whether OpenSSL or the underlaying Operating System (either a Unix or Win32 platform) is Year 2000 compliant is a different question which cannot be answered here.

- **What about mod_ssl and the Wassenaar Arrangement?** [L]

  First, let us explain what *Wassenaar* and its *Arrangement on Export Controls for Conventional Arms and Dual-Use Goods and Technologies* is: This is a international regime, established 1995, to control trade in conventional arms and dual-use goods and technology. It replaced the previous *CoCom* regime. 33 countries are signatories: Argentina, Australia, Austria, Belgium, Bulgaria, Canada, Czech Republic, Denmark, Finland, France, Germany, Greece, Hungary, Ireland, Italy, Japan, Luxembourg, Netherlands, New Zealand, Norway, Poland, Portugal, Republic of Korea, Romania, Russian Federation, Slovak Republic, Spain, Sweden, Switzerland, Turkey, Ukraine, United Kingdom and United States. For more details look at http://www.wassenaar.org/.

In short: The aim of the Wassenaar Arrangement is to prevent the build up of military capabilities that threaten regional and international security and stability. The Wassenaar Arrangement controls the export of cryptography as a dual-use good, i.e., one that has both military and civilian applications. However, the Wassenaar Arrangement also provides an exemption from export controls for mass-market software and free software.

In the current Wassenaar ``*List of Dual Use Goods and Technologies And Munitions*'', under ``*GENERAL SOFTWARE NOTE*'' (GSN) it says ``*The Lists do not control "software" which is either: 1. [...] 2. "in the public domain".*'' And under ``*DEFINITIONS OF TERMS USED IN THESE LISTS*'' one can find the definition: ``*"In the public domain": This means "technology" or "software" which has been made available without restrictions upon its further dissemination. N.B. Copyright restrictions do not remove "technology" or "software" from being "in the public domain".*''

So, both mod_ssl and OpenSSL are ``in the public domain'' for the purposes of the Wassenaar Agreement and its ``*List of Dual Use Goods and Technologies And Munitions List*''.

Additionally the Wassenaar Agreement itself has no direct consequence for exporting cryptography software. What is actually allowed or forbidden to be exported from the countries has still to be defined in the local laws of each country. And at least according to official press releases from the German BMWi (see here) and the Switzerland Bawi (see here) there will be no forthcoming export restriction for free cryptography software for their countries. Remember that mod_ssl is created in Germany and distributed from Switzerland.

So, mod_ssl and OpenSSL are not affected by the Wassenaar Agreement.

About Installation

- **When I access my website the first time via HTTPS I get a core dump?** [L]

  There can be a lot of reasons why a core dump can occur, of course. Ranging from buggy third-party modules, over buggy vendor libraries up to a buggy mod_ssl version. But the above situation is often caused by old or broken vendor DBM libraries. To solve it either build mod_ssl with the built-in SDBM library (specify `--enable-rule=SSL_SDBM` at the APACI command line) or switch from ``` ``SSLSessionCache dbm:'' ``` to the newer ``` ``SSLSessionCache shm:'' ``` variant (after you have rebuilt Apache with MM, of course).

- **My Apache dumps core when I add both mod_ssl and PHP3?** [L]

  Make sure you add mod_ssl to the Apache source tree first and then do a fresh configuration and installation of PHP3. For SSL support EAPI patches are required which have to change internal Apache structures. PHP3 needs to know about these in order to work correctly. Always make sure that `-DEAPI` is contained in the compiler flags when PHP3 is built.

- **When I startup Apache I get errors about undefined symbols like ap_global_ctx?** [L]

  This actually means you installed mod_ssl as a DSO, but without rebuilding Apache with EAPI. Because EAPI is a requirement for mod_ssl, you need an extra patched Apache (containing the EAPI patches) and you have to build this Apache with EAPI enabled (explicitly specify `--enable-rule=EAPI` at the APACI command line).

- **When I startup Apache I get permission errors related to SSLMutex?** [L]

  When you receive entries like ``` ``mod_ssl: Child could not open SSLMutex lockfile /opt/apache/logs/ssl_mutex.18332 (System error follows) [...] System: Permission denied (errno: 13)'' ``` this is usually caused by to restrictive permissions on the *parent* directories. Make sure that all parent directories (here `/opt`, `/opt/apache` and `/opt/apache/logs`) have the x-bit set at least for the UID under which Apache's children are running (see the `User` directive of Apache).

- **When I use the MM library and the shared memory cache each process grows 1.5MB according to `top' although I specified 512000 as the cache size?** [L]

  The additional 1MB are caused by the global shared memory pool EAPI allocates for all modules and which is not used by mod_ssl for various reasons. So the actually allocated shared memory is always 1MB more than what you specify on `SSLSessionCache`. But don't be confused by the display of `top': although is indicates that *each* process grow, this is not reality, of course. Instead the additional memory consumption is shared by all processes, i.e. the 1.5MB are allocated only once per Apache instance and not once per Apache server process.

- **Apache creates files in a directory declared by the internal EAPI_MM_CORE_PATH define. Is there a way to override the path using a configuration directive?** [L]

  No, there is not configuration directive, because for technical bootstrapping reasons, a directive not possible at all. Instead use

``CFLAGS='-DEAPI_MM_CORE_PATH="/path/to/wherever/"' ./configure ..." when building Apache or use option **-d** when starting `httpd`.

- **When I fire up the server, mod_ssl stops with the error "Failed to generate temporary 512 bit RSA private key", why? And a "PRNG not seeded" error occurs if I try "make certificate".** [L]

  Cryptographic software needs a source of unpredictable data to work correctly. Many open source operating systems provide a "randomness device" that serves this purpose (usually named `/dev/random`). On other systems, applications have to seed the OpenSSL Pseudo Random Number Generator (PRNG) manually with appropriate data before generating keys or performing public key encryption. As of version 0.9.5, the OpenSSL functions that need randomness report an error if the PRNG has not been seeded with at least 128 bits of randomness. So mod_ssl has to provide enough entropy to the PRNG to work correctly. For this one has to use the `SSLRandomSeed` directives (to solve the run-time problem) and create a `$HOME/.rnd` file to make sure enough entropy is available also for the "`make certificate`" step (in case the "`make certificate`" procedure is not able to gather enough entropy theirself by searching for system files).

About Configuration

- **Is it possible to provide HTTP and HTTPS with a single server?** [L]

  Yes, HTTP and HTTPS use different server ports, so there is no direct conflict between them. Either run two separate server instances (one binds to port 80, the other to port 443) or even use Apache's elegant virtual hosting facility where you can easily create two virtual servers which Apache dispatches: one responding to port 80 and speaking HTTP and one responding to port 443 speaking HTTPS.

- **I know that HTTP is on port 80, but where is HTTPS?** [L]

  You can run HTTPS on any port, but the standards specify port 443, which is where any HTTPS compliant browser will look by default. You can force your browser to look on a different port by specifying it in the URL like this (for port 666): `https://secure.server.dom:666/`

- **How can I speak HTTPS manually for testing purposes?** [L]

  While you usually just use

  ```
  $ telnet localhost 80
  GET / HTTP/1.0
  ```

  for simple testing the HTTP protocol of Apache, it's not so easy for HTTPS because of the SSL protocol between TCP and HTTP. But with the help of OpenSSL's `s_client` command you can do a similar check even for HTTPS:

  ```
  $ openssl s_client -connect localhost:443 -state -debug
  GET / HTTP/1.0
  ```

  Before the actual HTTP response you receive detailed information about the SSL handshake. For a more general command line client which directly understands both the HTTP and HTTPS scheme, can perform GET and POST methods, can use a proxy, supports byte ranges, etc. you should have a look at nifty cURL tool. With it you can directly check if your Apache is running fine on Port 80 and 443 as following:

  ```
  $ curl http://localhost/
  $ curl https://localhost/
  ```

- **Why does the connection hang when I connect to my SSL-aware Apache server?** [L]

  Because you connected with HTTP to the HTTPS port, i.e. you used an URL of the form ``http://" instead of ``https://". This also happens the other way round when you connect via HTTPS to a HTTP port, i.e. when you try to use ``https://" on a server that doesn't support SSL (on this port). Make sure you are connecting to a virtual server that supports SSL, which is probably the IP associated with your hostname, not localhost (127.0.0.1).

- **Why do I get ``Connection Refused'' messages when trying to access my freshly installed Apache+mod_ssl server via HTTPS?** [L]

  There can be various reasons. Some of the common mistakes is that people start Apache with just ``apachectl start" (or ``httpd") instead of ``apachectl startssl" (or ``httpd -DSSL". Or you're configuration is not correct. At least make sure that your ``Listen" directives match your ``<VirtualHost>" directives. And if all fails, please do yourself a favor and start over with the default configuration mod_ssl provides you.

- **In my CGI programs and SSI scripts the various documented `SSL_XXX` variables do not exist. Why?** [L]

Just make sure you have ``SSLOptions +StdEnvVars" enabled for the context of your CGI/SSI requests.

- **How can I use relative hyperlinks to switch between HTTP and HTTPS?** [L]

  Usually you have to use fully-qualified hyperlinks because you have to change the URL scheme. But with the help of some URL manipulations through mod_rewrite you can achieve the same effect while you still can use relative URLs:

  ```
  RewriteEngine on
  RewriteRule   ^/(.*):SSL$   https://%{SERVER_NAME}/$1 [R,L]
  RewriteRule   ^/(.*):NOSSL$ http://%{SERVER_NAME}/$1  [R,L]
  ```

  This rewrite ruleset lets you use hyperlinks of the form

  ```
  <a href="document.html:SSL">
  ```

About Certificates

- **What are RSA Private Keys, CSRs and Certificates?** [L]

  The RSA private key file is a digital file that you can use to decrypt messages sent to you. It has a public component which you distribute (via your Certificate file) which allows people to encrypt those messages to you. A Certificate Signing Request (CSR) is a digital file which contains your public key and your name. You send the CSR to a Certifying Authority (CA) to be converted into a real Certificate. A Certificate contains your RSA public key, your name, the name of the CA, and is digitally signed by your CA. Browsers that know the CA can verify the signature on that Certificate, thereby obtaining your RSA public key. That enables them to send messages which only you can decrypt. See the Introduction chapter for a general description of the SSL protocol.

- **Seems like there is a difference on startup between the original Apache and an SSL-aware Apache?** [L]

  Yes, in general, starting Apache with a built-in mod_ssl is just like starting an unencumbered Apache, except for the fact that when you have a pass phrase on your SSL private key file. Then a startup dialog pops up asking you to enter the pass phrase.

  To type in the pass phrase manually when starting the server can be problematic, for instance when starting the server from the system boot scripts. As an alternative to this situation you can follow the steps below under ``How can I get rid of the pass-phrase dialog at Apache startup time?".

- **How can I create a dummy SSL server Certificate for testing purposes?** [L]

  A Certificate does not have to be signed by a public CA. You can use your private key to sign the Certificate which contains your public key. You can install this Certificate into your server, and people using Netscape Navigator (not MSIE) will be able to connect after clicking OK to a warning dialogue. You can get MSIE to work, and your customers can eliminate the dialogue, by installing that Certificate manually into their browsers.

  Just use the ``make certificate" command at the top-level directory of the Apache source tree right before installing Apache via ``make install". This creates a self-signed SSL Certificate which expires after 30 days and isn't encrypted (which means you don't need to enter a pass-phrase at Apache startup time).

  BUT REMEMBER: YOU REALLY HAVE TO CREATE A REAL CERTIFICATE FOR THE LONG RUN! HOW THIS IS DONE IS DESCRIBED IN THE NEXT ANSWER.

- **Ok, I've got my server installed and want to create a real SSL server Certificate for it. How do I do it?** [L]

  Here is a step-by-step description:

  1. Make sure OpenSSL is really installed and in your `PATH`. But some commands even work ok when you just run the ``openssl" program from within the OpenSSL source tree as ``./apps/openssl".
  2. Create a RSA private key for your Apache server (will be Triple-DES encrypted and PEM formatted):

     ```
     $ openssl genrsa -des3 -out server.key 1024
     ```

     Please backup this `server.key` file and remember the pass-phrase you had to enter at a secure location. You can see the details of this RSA private key via the command:

     ```
     $ openssl rsa -noout -text -in server.key
     ```

And you could create a decrypted PEM version (not recommended) of this RSA private key via:

```
$ openssl rsa -in server.key -out server.key.unsecure
```

3. Create a Certificate Signing Request (CSR) with the server RSA private key (output will be PEM formatted):

```
$ openssl req -new -key server.key -out server.csr
```

Make sure you enter the FQDN ("Fully Qualified Domain Name") of the server when OpenSSL prompts you for the "CommonName", i.e. when you generate a CSR for a website which will be later accessed via `https://www.foo.dom/`, enter "www.foo.dom" here. You can see the details of this CSR via the command

```
$ openssl req -noout -text -in server.csr
```

4. You now have to send this Certificate Signing Request (CSR) to a Certifying Authority (CA) for signing. The result is then a real Certificate which can be used for Apache. Here you have two options: First you can let the CSR sign by a commercial CA like Verisign or Thawte. Then you usually have to post the CSR into a web form, pay for the signing and await the signed Certificate you then can store into a server.crt file. For more information about commercial CAs have a look at the following locations:

   - Verisign
     http://digitalid.verisign.com/server/apacheNotice.htm
   - Thawte Consulting
     http://www.thawte.com/certs/server/request.html
   - CertiSign Certificadora Digital Ltda.
     http://www.certisign.com.br
   - IKS GmbH
     http://www.iks-jena.de/produkte/ca/
   - Uptime Commerce Ltd.
     http://www.uptimecommerce.com
   - BelSign NV/SA
     http://www.belsign.be

Second you can use your own CA and now have to sign the CSR yourself by this CA. Read the next answer in this FAQ on how to sign a CSR with your CA yourself. You can see the details of the received Certificate via the command:

```
$ openssl x509 -noout -text -in server.crt
```

5. Now you have two files: `server.key` and `server.crt`. These now can be used as following inside your Apache's `httpd.conf` file:

```
       SSLCertificateFile    /path/to/this/server.crt
       SSLCertificateKeyFile /path/to/this/server.key
```

The `server.csr` file is no longer needed.

- **How can I create and use my own Certificate Authority (CA)?** [L]

The short answer is to use the `CA.sh` or `CA.pl` script provided by OpenSSL. The long and manual answer is this:

1. Create a RSA private key for your CA (will be Triple-DES encrypted and PEM formatted):

```
$ openssl genrsa -des3 -out ca.key 1024
```

Please backup this `ca.key` file and remember the pass-phrase you currently entered at a secure location. You can see the details of this RSA private key via the command

```
$ openssl rsa -noout -text -in ca.key
```

And you can create a decrypted PEM version (not recommended) of this private key via:

```
$ openssl rsa -in ca.key -out ca.key.unsecure
```

2. Create a self-signed CA Certificate (X509 structure) with the RSA key of the CA (output will be PEM formatted):

```
$ openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

You can see the details of this Certificate via the command:

```
$ openssl x509 -noout -text -in ca.crt
```

3. Prepare a script for signing which is needed because the ``openssl ca'' command has some strange requirements and the default OpenSSL config doesn't allow one easily to use ``openssl ca'' directly. So a script named `sign.sh` is distributed with the mod_ssl distribution (subdir `pkg.contrib/`). Use this script for signing.
4. Now you can use this CA to sign server CSR's in order to create real SSL Certificates for use inside an Apache webserver (assuming you already have a `server.csr` at hand):

   ```
   $ ./sign.sh server.csr
   ```

   This signs the server CSR and results in a `server.crt` file.

- **How can I change the pass-phrase on my private key file?** [L]

  You simply have to read it with the old pass-phrase and write it again by specifying the new pass-phrase. You can accomplish this with the following commands:

  ```
  $ openssl rsa -des3 -in server.key -out server.key.new
  $ mv server.key.new server.key
  ```

  Here you're asked two times for a PEM pass-phrase. At the first prompt enter the old pass-phrase and at the second prompt enter the new pass-phrase.

- **How can I get rid of the pass-phrase dialog at Apache startup time?** [L]

  The reason why this dialog pops up at startup and every re-start is that the RSA private key inside your server.key file is stored in encrypted format for security reasons. The pass-phrase is needed to be able to read and parse this file. When you can be sure that your server is secure enough you perform two steps:

  1. Remove the encryption from the RSA private key (while preserving the original file):

     ```
     $ cp server.key server.key.org
     $ openssl rsa -in server.key.org -out server.key
     ```

  2. Make sure the server.key file is now only readable by root:

     ```
     $ chmod 400 server.key
     ```

  Now `server.key` will contain an unencrypted copy of the key. If you point your server at this file it will not prompt you for a pass-phrase. HOWEVER, if anyone gets this key they will be able to impersonate you on the net. PLEASE make sure that the permissions on that file are really such that only root or the web server user can read it (preferably get your web server to start as root but run as another server, and have the key readable only by root).

  As an alternative approach you can use the ``SSLPassPhraseDialog exec:/path/to/program'' facility. But keep in mind that this is neither more nor less secure, of course.

- **How do I verify that a private key matches its Certificate?** [L]

  The private key contains a series of numbers. Two of those numbers form the "public key", the others are part of your "private key". The "public key" bits are also embedded in your Certificate (we get them from your CSR). To check that the public key in your cert matches the public portion of your private key, you need to view the cert and the key and compare the numbers. To view the Certificate and the key run the commands:

  ```
  $ openssl x509 -noout -text -in server.crt
  $ openssl rsa -noout -text -in server.key
  ```

  The `modulus' and the `public exponent' portions in the key and the Certificate must match. But since the public exponent is usually 65537 and it's bothering comparing long modulus you can use the following approach:

  ```
  $ openssl x509 -noout -modulus -in server.crt | openssl md5
  $ openssl rsa -noout -modulus -in server.key | openssl md5
  ```

  And then compare these really shorter numbers. With overwhelming probability they will differ if the keys are different. BTW, if I want to check to which key or certificate a particular CSR belongs you can compute

  ```
  $ openssl req -noout -modulus -in server.csr | openssl md5
  ```

- **What does it mean when my connections fail with an "alert bad certificate" error?** [L]

  Usually when you see errors like ``OpenSSL: error:14094412: SSL routines:SSL3_READ_BYTES:sslv3 alert bad certificate'' in the SSL logfile, this means that the browser was unable to handle the server certificate/private-key which perhaps contain a RSA-key not equal to 1024 bits. For instance Netscape Navigator 3.x is one of those browsers.

- **Why does my 2048-bit private key not work?** [L]

  The private key sizes for SSL must be either 512 or 1024 for compatibility with certain web browsers. A keysize of 1024 bits is recommended because keys larger than 1024 bits are incompatible with some versions of Netscape Navigator and Microsoft Internet Explorer, and with other browsers that use RSA's BSAFE cryptography toolkit.

- **Why is client authentication broken after upgrading from SSLeay version 0.8 to 0.9?** [L]

  The CA certificates under the path you configured with `SSLCACertificatePath` are found by SSLeay through hash symlinks. These hash values are generated by the ``openssl x509 -noout -hash`' command. But the algorithm used to calculate the hash for a certificate has changed between SSLeay 0.8 and 0.9. So you have to remove all old hash symlinks and re-create new ones after upgrading. Use the `Makefile` mod_ssl placed into this directory.

- **How can I convert a certificate from PEM to DER format?** [L]

  The default certificate format for SSLeay/OpenSSL is PEM, which actually is Base64 encoded DER with header and footer lines. For some applications (e.g. Microsoft Internet Explorer) you need the certificate in plain DER format. You can convert a PEM file `cert.pem` into the corresponding DER file `cert.der` with the following command: **`$ openssl x509 -in cert.pem -out cert.der -outform DER`**

- **I try to install a Verisign certificate. Why can't I find neither the `getca` nor `getverisign` programs Verisign mentions?** [L]

  This is because Verisign has never provided specific instructions for Apache+mod_ssl. Rather they tell you what you should do if you were using C2Net's Stronghold (a commercial Apache based server with SSL support). The only thing you have to do is to save the certificate into a file and give the name of that file to the `SSLCertificateFile` directive. Remember that you need to give the key file in as well (see `SSLCertificateKeyFile` directive). For a better CA-related overview on SSL certificate fiddling you can look at Thawte's mod_ssl instructions.

- **Can I use the Server Gated Cryptography (SGC) facility (aka Verisign Global ID) also with mod_ssl?** [L]

  Yes, mod_ssl since version 2.1 supports the SGC facility. You don't have to configure anything special for this, just use a Global ID as your server certificate. The *step up* of the clients are then automatically handled by mod_ssl under run-time. For details please read the `README.GlobalID` document in the mod_ssl distribution.

- **After I have installed my new Verisign Global ID server certificate, the browsers complain that they cannot verify the server certificate?** [L]

  That is because Verisign uses an intermediate CA certificate between the root CA certificate (which is installed in the browsers) and the server certificate (which you installed in the server). You should have received this additional CA certificate from Verisign. If not, complain to them. Then configure this certificate with the `SSLCertificateChainFile` directive in the server. This makes sure the intermediate CA certificate is send to the browser and this way fills the gap in the certificate chain.

About SSL Protocol

- **Why do I get lots of random SSL protocol errors under heavy server load?** [L]

  There can be a number of reasons for this, but the main one is problems with the SSL session Cache specified by the `SSLSessionCache` directive. The DBM session cache is most likely the source of the problem, so trying the SHM session cache or no cache at all may help.

- **Why has my webserver a higher load now that I run SSL there?** [L]

  Because SSL uses strong cryptographic encryption and this needs a lot of number crunching. And because when you request a webpage via HTTPS even the images are transfered encrypted. So, when you have a lot of HTTPS traffic the load increases.

- **Often HTTPS connections to my server require up to 30 seconds for establishing the connection, although sometimes it works faster?** [L]

  Usually this is caused by using a `/dev/random` device for `SSLRandomSeed` which is blocking in read(2) calls if not enough entropy is available. Read more about this problem in the refernce chapter under `SSLRandomSeed`.

- **What SSL Ciphers are supported by mod_ssl?** [L]

  Usually just all SSL ciphers which are supported by the version of OpenSSL in use (can depend on the way you built OpenSSL). Typically this at least includes the following:

- RC4 with MD5
- RC4 with MD5 (export version restricted to 40-bit key)
- RC2 with MD5
- RC2 with MD5 (export version restricted to 40-bit key)
- IDEA with MD5
- DES with MD5
- Triple-DES with MD5

To determine the actual list of supported ciphers you can run the following command:

```
$ openssl ciphers -v
```

- **I want to use Anonymous Diffie-Hellman (ADH) ciphers, but I always get ``no shared cipher'' errors?** [L]

  In order to use Anonymous Diffie-Hellman (ADH) ciphers, it is not enough to just put ``ADH'' into your `SSLCipherSuite`. Additionally you have to build OpenSSL with ``-DSSL_ALLOW_ADH''. Because per default OpenSSL does not allow ADH ciphers for security reasons. So if you are actually enabling these ciphers make sure you are informed about the side-effects.

- **I always just get a 'no shared ciphers' error if I try to connect to my freshly installed server?** [L]

  Either you have messed up your `SSLCipherSuite` directive (compare it with the pre-configured example in `httpd.conf-dist`) or you have choosen the DSA/DH algorithms instead of RSA under "`make certificate`" and ignored or overseen the warnings. Because if you have choosen DSA/DH, then your server no longer speaks RSA-based SSL ciphers (at least not until you also configure an additional RSA-based certificate/key pair). But current browsers like NS or IE only speak RSA ciphers. The result is the "no shared ciphers" error. To fix this, regenerate your server certificate/key pair and this time choose the RSA algorithm.

- **Why can't I use SSL with name-based/non-IP-based virtual hosts?** [L]

  The reason is very technical. Actually it's some sort of a chicken and egg problem: The SSL protocol layer stays below the HTTP protocol layer and encapsulates HTTP. When an SSL connection (HTTPS) is established Apache/mod_ssl has to negotiate the SSL protocol parameters with the client. For this mod_ssl has to consult the configuration of the virtual server (for instance it has to look for the cipher suite, the server certificate, etc.). But in order to dispatch to the correct virtual server Apache has to know the `Host` HTTP header field. For this the HTTP request header has to be read. This cannot be done before the SSL handshake is finished. But the information is already needed at the SSL handshake phase. Bingo!

- **When I use Basic Authentication over HTTPS the lock icon in Netscape browsers still shows the unlocked state when the dialog pops up. Does this mean the username/password is still transmitted unencrypted?** [L]

  No, the username/password is already transmitted encrypted. The icon in Netscape browsers is just not really synchronized with the SSL/TLS layer (it toggles to the locked state when the first part of the actual webpage data is transferred which is not quite correct) and this way confuses people. The Basic Authentication facility is part of the HTTP layer and this layer is above the SSL/TLS layer in HTTPS. And before any HTTP data communication takes place in HTTPS the SSL/TLS layer has already done the handshake phase and switched to encrypted communication. So, don't get confused by this icon.

- **When I connect via HTTPS to an Apache+mod_ssl+OpenSSL server with Microsoft Internet Explorer (MSIE) I get various I/O errors. What is the reason?** [L]

  The first reason is that the SSL implementation in some MSIE versions has some subtle bugs related to the HTTP keep-alive facility and the SSL close notify alerts on socket connection close. Additionally the interaction between SSL and HTTP/1.1 features are problematic with some MSIE versions, too. You've to work-around these problems by forcing Apache+mod_ssl+OpenSSL to not use HTTP/1.1, keep-alive connections or sending the SSL close notify messages to MSIE clients. This can be done by using the following directive in your SSL-aware virtual host section:

  ```
  SetEnvIf User-Agent ".*MSIE.*" \
           nokeepalive ssl-unclean-shutdown \
           downgrade-1.0 force-response-1.0
  ```

  Additionally it is known some MSIE versions have also problems with particular ciphers. Unfortunately one cannot workaround these bugs only for those MSIE particular clients, because the ciphers are already used in the SSL handshake phase. So a MSIE-specific `SetEnvIf` doesn't work to solve these problems. Instead one has to do more drastic adjustments to the global parameters. But before you decide to do this, make sure your clients really have problems. If not, do not do this, because it affects all(!) your clients, i.e., also your non-MSIE clients.

  The next problem is that 56bit export versions of MSIE 5.x browsers have a broken SSLv3 implementation which badly interacts with OpenSSL versions greater than 0.9.4. You can either accept this and force your clients to upgrade their browsers, or you downgrade to OpenSSL 0.9.4 (hmmm), or you can decide to workaround it by accepting the drawback that your workaround will horribly affect also other browsers:

```
SSLProtocol all -SSLv3
```

This completely disables the SSLv3 protocol and lets those browsers work. But usually this is an even less acceptable workaround. A more reasonable workaround is to address the problem more closely and disable only the ciphers which cause trouble.

```
SSLCipherSuite ALL:!ADH:!EXPORT56:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP
```

This also lets the broken MSIE versions work, but only removes the newer 56bit TLS ciphers.

Another problem with MSIE 5.x clients is that they refuse to connect to URLs of the form `https://12.34.56.78/` (IP-addresses are used instead of the hostname), if the server is using the Server Gated Cryptography (SGC) facility. This can only be avoided by using the fully qualified domain name (FQDN) of the website in hyperlinks instead, because MSIE 5.x has an error in the way it handles the SGC negotiation.

And finally there are versions of MSIE which seem to require that an SSL session can be reused (a totally non standard-conforming behaviour, of course). Connection with those MSIE versions only work if a SSL session cache is used. So, as a work-around, make sure you are using a session cache (see `SSLSessionCache` directive).

- **When I connect via HTTPS to an Apache+mod_ssl server with Netscape Navigator I get I/O errors and the message "Netscape has encountered bad data from the server" What's the reason?** [L]

  The problem usually is that you had created a new server certificate with the same DN, but you had told your browser to accept forever the old server certificate. Once you clear the entry in your browser for the old certificate, everything usually will work fine. Netscape's SSL implementation is correct, so when you encounter I/O errors with Netscape Navigator it is most of the time caused by the configured certificates.

About Support

- **What information resources are available in case of mod_ssl problems?** [L]

  The following information resources are available. In case of problems you should search here first.

  1. *Answers in the User Manual's F.A.Q. List (this)*
     http://httpd.apache.org/docs-2.0/ssl/ssl_faq.html
     First look inside the F.A.Q. (this text), perhaps your problem is such popular that it was already answered a lot of times in the past.
  2. *Postings from the modssl-users Support Mailing List* http://www.modssl.org/support/
     Second search for your problem in one of the existing archives of the modssl-users mailing list. Perhaps your problem popped up at least once for another user, too.
  3. *Problem Reports in the Bug Database* http://www.modssl.org/support/bugdb/
     Third look inside the mod_ssl Bug Database. Perhaps someone else already has reported the problem.

- **What support contacts are available in case of mod_ssl problems?** [L]

  The following lists all support possibilities for mod_ssl, in order of preference, i.e. start in this order and do not pick the support possibility you just like most, please.

  1. *Write a Problem Report into the Bug Database*
     http://www.modssl.org/support/bugdb/
     This is the preferred way of submitting your problem report, because this way it gets filed into the bug database (it cannot be lost) *and* send to the modssl-users mailing list (others see the current problems and learn from answers).
  2. *Write a Problem Report to the modssl-users Support Mailing List*
     modssl-users @ modssl.org
     This is the second way of submitting your problem report. You have to subscribe to the list first, but then you can easily discuss your problem with both the author and the whole mod_ssl user community.
  3. *Write a Problem Report to the author*
     rse @ engelschall.com
     This is the last way of submitting your problem report. Please avoid this in your own interest because the author is really a very busy men. Your mail will always be filed to one of his various mail-folders and is usually not processed as fast as a posting on modssl-users.

- **What information and details I've to provide to the author when writing a bug report?** [L]

You have to at least always provide the following information:

- *Apache, mod_ssl and OpenSSL version information*
  The mod_ssl version you should really know. For instance, it's the version number in the distribution tarball. The Apache version can be determined by running ``httpd -v". The OpenSSL version can be determined by running ``openssl version". Alternatively when you have Lynx installed you can run the command ``lynx -mime_header http://localhost/ | grep Server" to determine all information in a single step.
- *The details on how you built and installed Apache+mod_ssl+OpenSSL*
  For this you can provide a logfile of your terminal session which shows the configuration and install steps. Alternatively you can at least provide the author with the APACI `configure" command line you used (assuming you used APACI, of course).
- *In case of core dumps please include a Backtrace*
  In case your Apache+mod_ssl+OpenSSL should really dumped core please attach a stack-frame ``backtrace" (see the next question on how to get it). Without this information the reason for your core dump cannot be found. So you have to provide the backtrace, please.
- *A detailed description of your problem*
  Don't laugh, I'm totally serious. I already got a lot of problem reports where the people not really said what's the actual problem is. So, in your own interest (you want the problem be solved, don't you?) include as much details as possible, please. But start with the essentials first, of course.

- **I got a core dump, can you help me?** [L]

  In general no, at least not unless you provide more details about the code location where Apache dumped core. What is usually always required in order to help you is a backtrace (see next question). Without this information it is mostly impossible to find the problem and help you in fixing it.

- **Ok, I got a core dump but how do I get a backtrace to find out the reason for it?** [L]

  Follow the following steps:

  1. Make sure you have debugging symbols available in at least Apache and mod_ssl. On platforms where you use GCC/GDB you have to build Apache+mod_ssl with ``OPTIM="-g -ggdb3"" to achieve this. On other platforms at least ``OPTIM="-g"" is needed.
  2. Startup the server and try to produce the core-dump. For this you perhaps want to use a directive like ``CoreDumpDirectory /tmp" to make sure that the core-dump file can be written. You then should get a /tmp/core or /tmp/httpd.core file. When you don't get this, try to run your server under an UID != 0 (root), because most "current" kernels do not allow a process to dump core after it has done a setuid() (unless it does an exec()) for security reasons (there can be privileged information left over in memory). Additionally you can run ``/path/to/httpd -X" manually to force Apache to not fork.
  3. Analyze the core-dump. For this run ``gdb /path/to/httpd /tmp/httpd.core" or a similar command has to run. In GDB you then just have to enter the ``bt" command and, voila, you get the backtrace. For other debuggers consult your local debugger manual. Send this backtrace to the author.

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

SSL/TLS Strong Encryption: Glossary

``I know you believe you understand what you think I said, but I am not sure you realize that what you heard is not what I meant.''
Richard Nixon

Authentication

The positive identification of a network entity such as a server, a client, or a user. In SSL context the server and client *Certificate* verification process.

Access Control

The restriction of access to network realms. In Apache context usually the restriction of access to certain *URLs*.

Algorithm

An unambiguous formula or set of rules for solving a problem in a finite number of steps. Algorithms for encryption are usually called *Ciphers*.

Certificate

A data record used for authenticating network entities such as a server or a client. A certificate contains X.509 information pieces about its owner (called the subject) and the signing *Certificate Authority* (called the issuer), plus the owner's public key and the signature made by the CA. Network entities verify these signatures using CA certificates.

Certification Authority (CA)

A trusted third party whose purpose is to sign certificates for network entities it has authenticated using secure means. Other network entities can check the signature to verify that a CA has authenticated the bearer of a certificate.

Certificate Signing Request (CSR)

An unsigned certificate for submission to a *Certification Authority*, which signs it with the *Private Key* of their CA *Certificate*. Once the CSR is signed, it becomes a real certificate.

Cipher

An algorithm or system for data encryption. Examples are DES, IDEA, RC4, etc.

Ciphertext

The result after a *Plaintext* passed a *Cipher*.

Configuration Directive

A configuration command that controls one or more aspects of a program's behavior. In Apache context these are all the command names in the first column of the configuration files.

CONNECT

A HTTP command for proxying raw data channels over HTTP. It can be used to encapsulate other protocols, such as the SSL protocol.

Digital Signature

An encrypted text block that validates a certificate or other file. A *Certification Authority* creates a signature by generating a hash of the *Public Key* embedded in a *Certificate*, then encrypting the hash with its own *Private Key*. Only the CA's public key can decrypt the signature, verifying that the CA has authenticated the network entity that owns the *Certificate*.

Export-Crippled

Diminished in cryptographic strength (and security) in order to comply with the United States' Export Administration Regulations (EAR). Export-crippled cryptographic software is limited to a small key size, resulting in *Ciphertext* which usually can be decrypted by brute force.

Fully-Qualified Domain-Name (FQDN)

The unique name of a network entity, consisting of a hostname and a domain name that can resolve to an IP address. For example, `www` is a hostname, `whatever.com` is a domain name, and `www.whatever.com` is a fully-qualified domain name.

HyperText Transfer Protocol (HTTP)

The HyperText Transport Protocol is the standard transmission protocol used on the World Wide Web.

HTTPS

The HyperText Transport Protocol (Secure), the standard encrypted communication mechanism on the World Wide Web. This is actually just HTTP over SSL.

Message Digest

A hash of a message, which can be used to verify that the contents of the message have not been altered in transit.

OpenSSL

The Open Source toolkit for SSL/TLS; see http://www.openssl.org/

Pass Phrase

The word or phrase that protects private key files. It prevents unauthorized users from encrypting them. Usually it's just the secret encryption/decryption key used for *Ciphers*.

Plaintext

The unencrypted text.

Private Key

The secret key in a *Public Key Cryptography* system, used to decrypt incoming messages and sign outgoing ones.

Public Key

The publically available key in a *Public Key Cryptography* system, used to encrypt messages bound for its owner and to decrypt signatures made by its owner.

Public Key Cryptography

The study and application of asymmetric encryption systems, which use one key for encryption and another for decryption. A corresponding pair of such keys constitutes a key pair. Also called Asymmetric Crypography.

Secure Sockets Layer (SSL)

A protocol created by Netscape Communications Corporation for general communication authentication and encryption over TCP/IP networks. The most popular usage is *HTTPS*, i.e. the HyperText Transfer Protocol (HTTP) over SSL.

Session

The context information of an SSL communication.

SSLeay

The original SSL/TLS implementation library developed by Eric A. Young <eay@aus.rsa.com>; see http://www.ssleay.org/

Symmetric Cryptography

The study and application of *Ciphers* that use a single secret key for both encryption and decryption operations.

Transport Layer Security (TLS)

The successor protocol to SSL, created by the Internet Engineering Task Force (IETF) for general communication authentication and encryption over TCP/IP networks. TLS version 1 and is nearly identical with SSL version 3.

Uniform Resource Locator (URL)

The formal identifier to locate various resources on the World Wide Web. The most popular URL scheme is `http`. SSL uses the scheme `https`

X.509

An authentication certificate scheme recommended by the International Telecommunication Union (ITU-T) which is used for SSL/TLS authentication.

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Authentication

Related Modules	Related Directives
mod_auth mod_access	Allow AuthGroupFile AuthName AuthType AuthUserFile Deny Options Require

Authentication

Authentication is any process by which you verify that someone is who they claim they are. Authorization is any process by which someone is allowed to be where they want to go, or to have information that they want to have.

Introduction

If you have information on your web site that is sensitive or intended for only a small group of people, the techniques in this article will help you make sure that the people that see those pages are the people that you wanted to see them.

This article covers the "standard" way of protecting parts of your web site that most of you are going to use.

The prerequisites

The directives discussed in this article will need to go either in your main server configuration file (typically in a <Directory> section), or in per-directory configuration files (`.htaccess` files).

If you plan to use `.htaccess` files, you will need to have a server configuration that permits putting authentication directives in these files. This is done with the `AllowOverride` directive, which specifies which directives, if any, may be put in per-directory configuration files.

Since we're talking here about authentication, you will need an `AllowOverride` directive like the following:

```
    AllowOverride AuthConfig
```

Or, if you are just going to put the directives directly in your main server configuration file, you will of course need to have write permission to that file.

And you'll need to know a little bit about the directory structure of your server, in order to know where some files are kept. This should not be terribly difficult, and I'll try to make this clear when we come to that point.

Getting it working

Here's the basics of password protecting a directory on your server.

You'll need to create a password file. This file should be placed somewhere not accessible from the web. This is so that folks cannot download the password file. For example, if your documents are served out of `/usr/local/apache/htdocs` you might want to put the password file(s) in `/usr/local/apache/passwd`.

To create the file, use the htpasswd utility that came with Apache. This be located in the `bin` directory of wherever you installed Apache. To create the file, type:

```
        htpasswd -c /usr/local/apache/passwd/password rbowen
```

`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
```

If `htpasswd` is not in your path, of course you'll have to type the full path to the file to get it to run. On my server, it's located at `/usr/local/apache/bin/htpasswd`

Next, you'll need to configure the server to request a password and tell the server which users are allowed access. You can do this either by editing the `httpd.conf` file or using an `.htaccess` file. For example, if you wish to protect the directory `/usr/local/apache/htdocs/secret`, you can use the following directives, either placed in the file `/usr/local/apache/htdocs/secret/.htaccess`, or placed in httpd.conf inside a <Directory /usr/local/apache/apache/htdocs/secret> section.

```
        AuthType Basic
        AuthName "Restricted Files"
        AuthUserFile /usr/local/apache/passwd/passwords
        require user rbowen
```

Let's examine each of those directives individually. The AuthType directive selects that method that is used to authenticate the user. The most common method is `Basic`, and this is the method implemented by mod_auth. It is important to be aware, however, that Basic authentication sends the password from the client to the browser unencrypted. This method should therefore not be used for highly sensitive data. Apache supports one other authentication method: `AuthType Digest`. This method is implemented by mod_auth_digest and is much more secure. Only the most recent versions of clients are known to support Digest authentication.

The AuthName directive sets the *Realm* to be used in the authentication. The realm serves two major functions. First, the client often presents this information to the user as part of the password dialog box. Second, it is used by the client to determine what password to send for a given authenticated area. So, for example, once a client has authenticated in the `"Restricted Files"` area, it will automatically retry the same password for any area on the same server that is marked with the `"Restricted Files"` Realm. Therefore, you can prevent a user from being prompted more than once for a password by letting multiple restricted areas share the same realm. Of course, for security reasons, the client will always need to ask again for the password whenever the hostname of the server changes.

The AuthUserFile directive sets the path to the password file that we just created with `htpasswd`. If you have a large number of users, it can be quite slow to search through a plain text file to authenticate the user on each request. Apache also has the ability to store user information in fast database files. The mod_auth_dbm module provides the AuthDBMUserFile directive. These files can be created and manipulated with the dbmmanage program. Many other types of authentication options are available from third party modules in the Apache Modules Database.

Finally, the require directive provides the authorization part of the process by setting the user that is allowed to access this region of the server. In the next section, we discuss various ways to use the `require` directive.

Letting more than one person in

The directives above only let one person (specifically someone with a username of `rbowen`) into the directory. In most cases, you'll want to let more than one person in. This is where the AuthGroupFile comes in.

If you want to let more than one person in, you'll need to create a group file that associates group names with a list of users in that group. The format of this file is pretty simple, and you can create it with your favorite editor. The contents of the file will look like this:

```
GroupName: rbowen dpitts sungo rshersey
```

That's just a list of the members of the group in a long line separated by spaces.

To add a user to your already existing password file, type:

```
htpasswd /usr/local/apache/passwd/password dpitts
```

You'll get the same response as before, but it will be appended to the existing file, rather than creating a new file. (It's the `-c` that makes it create a new password file.

Now, you need to modify your `.htaccess` file to look like the following:

```
AuthType Basic
AuthName "By Invitation Only"
AuthUserFile /usr/local/apache/passwd/passwords
AuthGroupFile /usr/local/apache/passwd/groups
require group GroupName
```

Now, anyone that is listed in the group `GroupName`, and has an entry in the `password` file, will be let in, if they type the correct password.

There's another way to let multiple users in that is less specific. Rather than creating a group file, you can just use the following directive:

```
require valid-user
```

Using that rather than the `require user rbowen` line will allow anyone in that is listed in the password file, and who correctly enters their password. You can even emulate the group behavior here, by just keeping a separate password file for each group. The advantage of this approach is that Apache only has to check one file, rather than two. The disadvantage is that you have to maintain a bunch of password files, and remember to reference th right one in the `AuthUserFile` directive.

Possible problems

Because of the way that Basic authentication is specified, your username and password must be verified every time you request a document from the server. This is even if you're reloading the same page, and for every image on the page (if they come from a protected directory). As you can imagine, this slows things down a little. The amount that it slows things down is proportional to the size of the password file, because it has to open up that file, and go down the list of users until it gets to your name. And it has to do this every time a page is loaded.

A consequence of this is that there's a practical limit to how many users you can put in one password file. This limit will vary depending on the performance of your particular server machine, but you can expect to see slowdowns once you get above a few hundred entries, and may wish to consider a different authentication method at that time.

What other neat stuff can I do?

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.

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 `order` directive goes hand-in-hand with these two, and 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); you may provide multiple addresses or domain names, if desired.

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 covered by this directive. If, instead, you have a machine name, rather than an IP address, you can use that.

```
        deny from host.example.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 moreidiots.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 dev.example.com
```

Listing just the `allow` directive would not do what you want, because it will let folks from that host in, in addition to letting everyone in. What you want is to let *only* those folks in.

More information

You should also read the documentation for `mod_auth` and `mod_access` which contain some more information about how this all works.

Apache HTTP Server Version 2.0

Dynamic Content with CGI

Dynamic Content with CGI

Related Modules	Related Directives
mod_alias mod_cgi	AddHandler 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 to 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://dev.rcbowen.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 if 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 a 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 2.0

Apache HTTP Server Version 2.0

Apache Tutorial: Introduction to Server Side Includes

Apache Tutorial: Introduction to Server Side Includes

Related Modules	Related Directives
mod_include mod_cgi mod_expires	Options XBitHack AddType SetOutputFilter BrowserMatchNoCase

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
        AddOutputFilter INCLUDES .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 `strftime`. 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 2.0

Apache HTTP Server Version 2.0

Apache Tutorials

Warning: This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

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 2.0

Apache HTTP Server Version 2.0

Using Apache with Microsoft Windows

This document explains how to install, configure and run Apache 2.0 under Microsoft Windows. If you find any bugs, or wish to contribute in other ways, please use our bug reporting page.

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.

At this time, support for Windows 95, 98 and ME is incomplete. Apache 2.0 is not expected to work on those platforms at this time. If you are interested in helping with that effort, please see the developer's site for information on how to get involved. Support will likely be provided at some point in the future, and patches to allow Apache to work on 95, 98 and ME are welcome!

Requirements

Apache 2.0 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, 98 and ME, but these are not tested, and are never recommended for production servers. In all cases TCP/IP networking must be installed.

If running on Windows 95, the "Winsock2" upgrade MUST BE INSTALLED. "Winsock2" for Windows 95 is available here.

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 later Service Packs.

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 version of Apache for Windows with the `.msi` extension. This is a single Microsoft Installer file containing Apache, ready to install and run. There is a seperate `.zip` file containing _only_ the source code, to compile Apache yourself with the Microsoft Visual C++ (Visual Studio) tools.

Installing Apache for Windows

Run the Apache .msi file you downloaded above. This will ask for:

- the directory to install Apache into (the default is `\Program Files\Apache Group\Apache` although you can change this to any other directory)
- the start menu name (default is "Apache Web Server")
- the installation type. The "Typical" option installs everything except the source code. The "Minimum" option does not install the manuals or source code. Choose the "Custom" install if you want to install the source code.

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. 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.conf.default. After the installation has finished you should manually check to see what is in new in the .default 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 a 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.

Running Apache for Windows

There are two ways you can run Apache:

- As a "service" (available on Windows NT/2000, or a pseudo-service on Windows 95, 98 or ME). This is the best option if you want Apache to automatically start when you machine boots, and to keep Apache running when you log-off.
- From a console window. This MUST be used by any administrator to test before to attempting to run as a 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 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 Signalling Console Apache when Running for how to control Apache from the command line.

If the Apache console window closes immediately (or unexpectedly), 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 change to the logs folder, and 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
  cd logs
  more <error.log
```

Complete the steps above before you proceed to attempt to start Apache as a Window NT/2000 service!

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", run the "Install Apache as Service (NT only)" option from the Start menu. Once this is done you can start the "Apache" service by opening the Services window (in the Control Panel), selecting Apache, then clicking on Start. Apache will now be running 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 control line with:

```
  NET START APACHE
  NET STOP APACHE
```

See Signalling Service Apache when Running for more information on installing and controlling Apache services.

Apache, unlike many other Windows NT/2000 services, logs any errors to its own error.log file in the logs folder within the Apache server root folder. You will *not* find Apache error details in the Windows NT Event Log.

After starting Apache running (either in a console window or as a service) if will be listening to port 80 (unless you changed the Listen directive 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. Again, if you change the configuration of the Windows NT/2000 service for Apache, first attempt to start it from the command line to assure that the service starts with no errors.

Because Apache *CANNOT* share the same port with another TCPIP 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 TCPIP ports.

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.

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 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. **Warning: The server configuration file is reread when the new child process is started. If you have modified httpd.conf, the new child may not start or you may receive unexpected results.**

  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 contains 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 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.

- 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 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'.

You can install Apache as a Windows NT service as follows:

```
apache -k install -n "service name"
```

To install a service to use a particular configuration, specify the configuration file when the service is installed:

```
apache -k install -n "service name" -f "\my server\conf\my.conf"
```

To remove an Apache service, use

```
apache -k uninstall -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:

```
apache -n "service name" -t
```

To start a console Apache using a service's configuration file:

```
apache -n "service name"
```

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 Rpcss entry to the list.

Running Apache for Windows from the Command Line

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.
- 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 version-specific registry key is created in the Windows registry:

```
HKEY_LOCAL_MACHINE\Software\Apache Group\Apache\1.3.7

HKEY_LOCAL_MACHINE\Software\Apache Group\Apache\2.0a3
```

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 that 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.

Signalling Service Apache when running

On Windows NT, multiple instances of Apache can be run as services. Signal an Apache service to start, restart, or shutdown as follows:

```
apache -n "service name" -k start
apache -n "service name" -k restart
```

```
    apache -n "service name" -k shutdown
```

In addition, you can use the native NT NET command to start and stop Apache services as follows:

```
    NET START "service name"
    NET STOP "service name"
```

Signalling Console Apache when running

On Windows 95, Apache runs as a console application. You can tell a running Apache to stop by opening another console window and typing:

```
    apache -k shutdown
```

This should be used instead of pressing Control-C in the running Apache console window, because it lets Apache end any current transactions and cleanup gracefully.

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 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 Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Compiling Apache for Microsoft Windows

There are many important points before you begin compiling Apache. See Using Apache with Microsoft Windows before you begin.

Requirements

Compiling Apache requires the following environment to be properly installed;

- Disk Space

  Make sure you have at least 50 MB of free disk space available. After installation Apache requires approximately 10 MB of disk space, plus space for log and cache files, which can grow rapidly. The actual disk space requirements will vary considerably based on your chosen configuration and any third-party modules or libraries.

- Microsoft Visual C++ 5.0 or higher.

  Apache can be built using the command line tools, or from within the Visual Studio IDE Workbench. The command line build requires the environment to reflect the PATH, INCLUDE, LIB and other variables that can be configured with the vcvars32 batch file:

  ```
  "c:\Program Files\DevStudio\VC\Bin\vcvars32.bat"
  ```
- The Windows Platform SDK.

  Visual C++ 5.0 builds require an updated Microsoft Windows Platform SDK to enable some Apache features. For command line builds, the Platform SDK environment is prepared by the setenv batch file:

  ```
  "c:\Program Files\Platform SDK\setenv.bat"
  ```
  The Platform SDK files distributed with Visual C++ 6.0 and later are sufficient, so users of later version may skip this requirement.

  Note that the Windows Platform SDK update is required to enable all supported mod_isapi features. 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/downloads/sdks/platform/platform.asp.

- The awk utility (awk, gawk or similar.)

  To install Apache within the build system, several files are modified using the awk.exe utility. awk was chosen since it is a very small download (compared with Perl or WSH/VB) and accomplishes the task of generating files. Brian Kernighan's http://cm.bell-labs.com/cm/cs/who/bwk/ site has a compiled native Win32 binary, http://cm.bell-labs.com/cm/cs/who/bwk/awk95.exe which you must save with the name awk.exe rather than awk95.exe.

  Note that Developer Studio IDE will only find awk.exe from the Tools menu Options... Directories tab list of Executable file paths. Add the path for awk.exe to this list, and your system PATH environment variable, as needed.

- [Optional] OpenSSL libraries (for mod_ssl and ab.exe with ssl support)

  Caution: there are significant restrictions and prohibitions on the use and distribution of strong cryptography and patented intellectual property throughout the world. OpenSSL includes strong cryptography controlled by both export regulations and domestic law, as well as intellectual property protected by patent, in the United States and elsewhere. Neither the Apache Software Foundation nor the OpenSSL project can provide legal advise regarding possession, use, or distribution of the code provided by the OpenSSL project. **Consult your own legal counsel, you are responsible for your own actions.**

  OpenSSL must be installed into a srclib subdirectory named openssl, obtained from http://www.openssl.org/source/, in order to compile mod_ssl or the abs project (ab.exe with SSL support.) To prepare OpenSSL for both release and debug builds of Apache, and disable the patent protected features in 0.9.6c (as built by the ASF for binary distribution from the United States), you might use the following build commands;

```
perl util\mkfiles.pl >MINFO
perl util\mk1mf.pl dll no-asm no-mdc2 no-rc5 no-idea VC-WIN32 >makefile
perl util\mk1mf.pl dll debug no-asm no-mdc2 no-rc5 no-idea VC-WIN32 >makefile.dbg
perl util\mkdef.pl 32 libeay no-asm no-mdc2 no-rc5 no-idea >ms\libeay32.def
perl util\mkdef.pl 32 ssleay no-asm no-mdc2 no-rc5 no-idea >ms\ssleay32.def
nmake
nmake -f makefile.dbg
```

- [Optional] zlib sources (for mod_deflate)

  Zlib must be installed into a srclib subdirectory named zlib, however those sources need not be compiled. The build system will compile the compression sources directly into the mod_deflate module. Zlib can be obtained from http://www.gzip.org/zlib/ -- mod_deflate is confirmed to build correctly with version 1.1.4.

Command-Line Build

First, unpack the Apache distribution into an appropriate directory. Open a command-line prompt and cd to that directory.

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 to compiled the release or debug build, respectively:

```
nmake /f Makefile.win _apacher

nmake /f Makefile.win _apached
```

Either command will compile Apache. The latter will include debugging information in the resulting files, making it easier to find bugs and track down problems.

Developer Studio Workspace IDE Build

Apache can also be compiled using VC++'s VisualStudio development environment. To simplify this process, a VisualStudio workspace, Apache.dsw, is provided. 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.

Open the Apache.dsw workspace, and select InstallBin (Release or Debug build, as desired) as the Active Project. InstallBin causes all related project to be built, and then invokes Makefile.win to move the compiled executables and dlls. You may personalize the INSTDIR= choice by changing InstallBin's Settings, General tab, Build command line entry. INSTDIR defaults to the /Apache2 directory. If you only want a test compile (without installing) you may build the BuildBin project instead.

The .dsp project files are distributed in Visual C++ 6.0 format. Visual C++ 5.0 (97) will recognize them. Visual C++ 7.0 (.net) must convert Apache.dsw plus the .dsp files into an .msproj format, be sure you reconvert the .msproj file if any of the source .dsp files change!

Exported .mak files pose a greater hassle, but they are required for Visual C++ 5.0 and 7.0 users to build mod_ssl, ab with SSL support or mod_deflate, since only VC 6.0 knows how to invoke .dsp files directly. Build the entire project from within the IDE, then export all makefiles. You must build the projects in order to create all dynamic auto-generated targets, so that dependencies can be parsed correctly. Run the following command to fix the paths so they will build anywhere;

```
perl srclib\apr\build\fixwin32mak.pl
```

You must type this command from the *top level* directory of the httpd source tree. Every .mak and .dep project file within the current directory and below will be converted, and the timestamps adjusted to reflect the .dsp. If you contribute back a patch that revises project files, you must submit project files in Visual Studio 6.0 format.

Project Components

The Apache.dsw workspace and makefile.win nmake script both build the .dsp projects of the Apache server in the following sequence:

1. `srclib\apr\apr.dsp`
2. `srclib\apr\libapr.dsp`
3. `srclib\apr-util\uri\gen_uri_delims.dsp`

4. `srclib\apr-util\xml\expat\lib\xml.dsp`
5. `srclib\apr-util\aprutil.dsp`
6. `srclib\apr-util\libaprutil.dsp`
7. `srclib\pcre\dftables.dsp`
8. `srclib\pcre\pcre.dsp`
9. `srclib\pcre\pcreposix.dsp`
10. `server\gen_test_char.dsp`
11. `libhttpd.dsp`
12. `Apache.dsp`

In addition, the `modules\` subdirectory tree contains project files for the majority of the modules.

The `support\` directory contains project files for additional programs that are not part of the Apache runtime, but are used by the administrator to test Apache and maintain password and log files. Windows-specific support projects are broken out in the `support\win32\` directory.

1. `support\ab.dsp`
2. `support\htdigest.dsp`
3. `support\htpasswd.dsp`
4. `support\logresolve.dsp`
5. `support\rotatelogs.dsp`
6. `support\win32\ApacheMonitor.dsp`
7. `support\win32\wintty.dsp`

Once Apache has been compiled, it needs to be installed in its server root directory. The default is the `\Apache2` directory, of the same drive.

To build and install all the files into the desired folder *dir* automatically, use one the following nmake commands:

```
nmake /f Makefile.win installr INSTDIR=dir

nmake /f Makefile.win installd INSTDIR=dir
```

The *dir* argument to INSTDIR gives the installation directory; it can be omitted if Apache is to be installed into \Apache2.

This will install the following:

- *dir*`\bin\Apache.exe` - Apache executable
- *dir*`\bin\ApacheMonitor.exe` - Service monitor taskbar icon utility
- *dir*`\bin\htdigest.exe` - Digest auth password file utility
- *dir*`\bin\htdbm.exe` - SDBM auth database password file utility
- *dir*`\bin\htpasswd.exe` - Basic auth password file utility
- *dir*`\bin\logresolve.exe` - Log file dns name lookup utility
- *dir*`\bin\rotatelogs.exe` - Log file cycling utility
- *dir*`\bin\wintty.exe` - Console window utility
- *dir*`\bin\libapr.dll` - Apache Portable Runtime shared library
- *dir*`\bin\libaprutil.dll` - Apache Utility Runtime shared library
- *dir*`\bin\libhttpd.dll` - Apache Core library
- *dir*`\modules\mod_*.so` - Loadable Apache modules
- *dir*`\conf` - Configuration directory
- *dir*`\logs` - Empty logging directory
- *dir*`\include` - C language header files
- *dir*`\lib` - Link library files

Warning about building Apache from the development tree

Note; only the .dsp files are maintained between release builds. The .mak files are NOT regenerated, due to the tremendous waste of reviewer's time. Therefore, you cannot rely on the NMAKE commands above to build revised .dsp project files unless you then export all .mak files yourself from the project. This is unnecessary if you build from within the Microsoft Developer Studio environment.

Note: it is very worthwhile to build the BuildBin target project (or the command line _apacher or _apached target) prior to exporting the make files. Many files are autogenerated in the build process. Only a full build provides all of the dependent files required to build proper dependency trees for correct build behavior.

In order to create distribution .mak files, always review the generated .mak (or .dep) dependencies for Platform SDK or other garbage includes. The DevStudio\SharedIDE\bin\ (VC5) or DevStudio\Common\MSDev98\bin\ (VC6) directory contains the sysincl.dat file, which must list all exceptions. Update this file (including both forward and backslashed paths, such as both sys/time.h and sys\time.h) to include such dependencies. Including local-install paths in a distributed .mak file will cause the build to fail completely. And don't forget to run srclib/apr/build/fixwin32mak.pl in order to fix absolute paths within the .mak files.

Apache HTTP Server Version 2.0

Running Apache for Windows as a Service

Apache can be run as a service on Windows NT/2000. (There is also some HIGHLY 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.

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", run the "Install Apache as Service (NT only)" option from the Start menu. Once this is done you can start the "Apache" service by opening the Services window (in the Control Panel), 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 any errors to its own error.log file in the logs folder within the Apache server root folder. You will *not* find Apache error details in the Windows NT 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 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'.

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"
```

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:

```
    apache -n "service name" -t
```

To start a console Apache using a service's configuration file:

```
    apache -n "service name"
```

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 Rpcss entry to the list.

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 privilages 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 privilages to the SYSTEM account! Create a new user account instead, grant the appropriate privilages to that user, and use the 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 privilages 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 Running Apache for Windows from the Command Line.

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 occured.

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!

In addition, you can use the native NT NET command to start and stop Apache services as follows:

```
    NET START "service name"
    NET STOP "service name"
```

Again, quotes are only required if the service name contains spaces.

HIGHLY 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 *highly experimental*, if it works (at all) the Apache Sofware 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 will display an error message which will be useful in tracking down the cause of the problem.

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 privilages. 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 Version 2.0

Apache HTTP Server Version 2.0

Using Apache With Novell NetWare

This document explains how to install, configure and run Apache 2.0 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 dev-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

Apache 2.0 is designed to run on NetWare 5.x and above.

If running on NetWare 5.0 you must install Service Pack 7 or above.

If running on NetWare 5.1 you must install Service Pack 4 or above.

If running on NetWare 6 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.

Follow these steps to install Apache on NetWare from the binary download (assuming you will install to sys:/apache2):

- 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:/APACHE2 to the search path. EXAMPLE: SEARCH ADD SYS:\APACHE2

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 `Apache2` on a NetWare volume
- Copy APACHE2.NLM, APRLIB.NLM, HTDIGEST.NLM, HTPASSWD.NLM to SYS:/APACHE2
- Create a directory under SYS:/APACHE2 called CONF
- Copy the HTTPD-STD.CONF file to the SYS:/APACHE2/CONF directory and rename to HTTPD.CONF
- Copy the MIME.TYPES and MAGIC files to SYS:/APACHE2/CONF directory
- Copy all files and subdirectories in \HTTPD-2.0\DOCS\ICONS to SYS:/APACHE2/ICONS
- Copy all files and subdirectories in \HTTPD-2.0\DOCS\MANUAL to SYS:/APACHE2/MANUAL
- Copy all files and subdirectories in \HTTPD-2.0\DOCS\ERROR to SYS:/APACHE2/ERROR
- Copy all files and subdirectories in \HTTPD-2.0\DOCS\DOCROOT to SYS:/APACHE2/HTDOCS
- Create the directory SYS:/APACHE2/LOGS on the server
- Create the directory SYS:/APACHE2/CGI-BIN on the server
- Create the directory SYS:/APACHE2/MODULES and copy all nlm modules into the modules directory
- Edit the HTTPD.CONF file searching for all @@*Value*@@ markers and replacing them with the appropriate setting
- Add SYS:/APACHE2 to the search path. EXAMPLE: SEARCH ADD SYS:\APACHE2

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 = apache2 apache2
```

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 Listen directive 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 apache2
  or
apache2 shutdown
```

If apache is running in a protected address space specify the address space in the unload statement:

```
unload address space = apache2 apache2
```

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

```
apache2 -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:/apache2". invoking apache with the -V switch will display this value labeled as HTTPD_ROOT.

Apache 2.0 for NetWare includes a set of command line directives that can be used to modify or display information about the running instance of the web server. Each of these directives must be preceded by the keyword APACHE2:

- RESTART - Instructs Apache to terminate all running worker threads as they become idle, reread the configuration file and restart each worker thread based on the new configuration.
- VERSION - Displays version information about the currently running instance of Apache.
- MODULES - Displays a list of loaded modules both built-in and external.
- DIRECTIVES - Displays a list of all available directives.
- SETTINGS - Enables or disables the thread status display on the console. When enabled, a status of the number of running threads is displayed along with their status.
- SHUTDOWN - Terminates the running instance of the Apache web server.

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 in some Unix implementations. Instead there are only threads running: a parent thread, and a multiple child threads which handle the requests. So the "process"-management directives are different:

  MaxRequestsPerChild - Like the Unix directive, this controls how many requests a worker thread will serve before exiting. The recommended default, `MaxRequestsPerChild 0`, causes the thread to continue servicing request indefinitely. It is recommended on NetWare, unless there is some specific reason, that this directive always remain set to 0.

  StartThreads - This directive tells the server how many threads it should start initially. The recommended default is `StartThreads 50`.

  MinSpareThreads - This directive instructs the server to spawn additional worker threads if the number of idle threads ever falls below this value. The recommended default is `MinSpareThreads 10`.

  MaxSpareThreads - This directive instructs the server to begin terminating worker threads if the number of idle threads ever exceeds this value. The recommended default is `MaxSpareThreads 100`.

  MaxThreads - This directive limits the total number of work threads to a maximum value. The recommended default is `ThreadsPerChild 250`.

  ThreadStackSize - This directive tells the server what size of stack to use for the individual worker thread. 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. It is recommended that all rooted file paths begin with a volume name. If omitted, Apache will assume the SYS: volume.

- 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 `\Apache2\modules` directory. To activate these, or other modules, the 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/status.nlm
  ```

Information on creating loadable modules is also available.

Compiling Apache for NetWare

Compiling Apache requires MetroWerks CodeWarrior 6.x or higher to be properly installed. Once Apache has been built, it needs to be installed on a NetWare volume's root directory. The default is the `sys:/Apache2` directory.

Before running the server you must fill out the conf directory. Copy the file HTTPD-STD.CONF from the distribution conf directory and rename it to HTTPD.CONF. Edit the HTTPD.CONF file searching for all @@*Value*@@ markers and replacing them with the appropriate setting. Copy over the conf/magic and conf/mime.types files as well.

Requirements:

The following development tools are required to build Apache 2.0 for NetWare:

- Metrowerks CodeWarrior 6.0 or higher with the NetWare PDK 3.0 or higher.
- NetWare Libraries for C (LibC)
- WinSock 2 Developer Components for NetWare
- To build using either the project file or the make files, requires an AWK utility (awk, gawk or similar). AWK can be downloaded from http://developer.novell.com/ndk/apache.htm. The utility must be found in your windows path and must be named awk.exe.
- To build using the makefiles, you will need GNU make version 3.78.1 (GMake) available at http://developer.novell.com/ndk/apache.htm.

Building Apache using the Metrowerks Project Files:

All major pieces of Apache and APR are built using the ApacheNW.mcp and LibAprNW.mcp project files. This includes modules such as status, info, proxy, etc.

- Set the environment variable "NovellLibC" to the location of the NetWare Libraries for C SDK (ex. Set NovellLibC=c:\novell\ndk\libc).
- Make sure that the path to the CodeWarrior command line tools (MWCCNLM.exe, MWLDNLM.exe) has been included in the system's PATH environment variable.
- Make sure that the path to the AWK utility has been included in the system's PATH environment variable.
- Download the source code and unzip to an appropriate directory on your workstation.
- Change directory to \httpd\srclib\apr\build and run the batch file prebuildnw.bat. The batch file will setup the build environment for building the APR libraries. It will also run 2 AWK scripts that will generate the export files for APR.
- Change directory to \httpd\srclib\apr and extract the project file LIBAPRNW.mcp from the LIBAPRNW.mcp.zip file.
- Open the LIBAPRNW.mcp project file in the Metrowerks IDE.
- Select the target "Build Util - Gen URL Delim" and build the target. This target will produce the NLM "GENURI.nlm"
- Copy the file GENURI.nlm to the SYS: volume of a NetWare server and run using the following command:
  - SYS:\genuri > sys:\uri_delims.h
- Copy the file "uri_delims.h" to the directory \httpd\srclib\apr-util\uri on the build machine.
- Select the target "APR Debug NLM" or "APR Release NLM" in the IDE and build. This will produce the file APRLIB.nlm.
  - *OPTIONAL*: Select any of the LIB targets to produce a statically linkable libraries.
- Change directory to \httpd\build and run the batch file prebuildnw.bat. This batch file will setup the build environment for building the APACHE.nlm. It will also run several AWK scripts that will generate the export files for APACHE.
- Change directory to \http and extract the project file ApacheNW.mcp from the ApacheNW.mcp.zip file.
- Open the ApacheNW.mcp project file in the Metrowerks IDE.
- Select the target "Build Utility - DFTables" and build the target.
- Select the target "Build Util - Gen Test Chars" and build the target.
- Copy the files "GENCHARS.nlm" and "DFTABLES.nlm" to the SYS: volume of a NetWare server and run using the following commands:
  - SYS:\genchars > sys:\test_char.h
  - SYS:\dftables > sys:\chartables.c
- Copy the files "test_char.h" and "chartables.c" to the directory \httpd\os\netware on the build machine.
- Select the target "Apache Full Debug" or "Apache Full Release" in the IDE and build. This will produce the file APACHE2.nlm along

with all of the external module NLMs.

Building Apache using the NetWare makefiles:

- Set the environment variable "NOVELLLIBC" to the location of the NetWare Libraries for C SDK (ex. Set NOVELLLIBC=c:\novell\ndk\libc).
- Set the environment variable "METROWERKS" to the location where you installed the Metrowerks CodeWarrior compiler (ex. Set METROWERKS=C:\Program Files\Metrowerks\CodeWarrior). If you installed to the default location C:\Program Files\Metrowerks\CodeWarrior, you don't need to set this.
- Set the environment variable "AP_WORK" to the full path of the \httpd directory.
- Set the environment variable "APR_WORK" to the full path of the \httpd\srclib\apr directory.
- Make sure that the path to the AWK utility and the GNU make utility (gmake.exe) have been included in the system's PATH environment variable.
- Download the source code and unzip to an appropriate directory on your workstation.
- Change directory to \httpd\srclib\apr-util\uri and build GENURI.nlm by running "gmake -f nwgnumakefile"
- Copy the file GENURI.nlm to the SYS: volume of a NetWare server and run using the following command:
    - SYS:\genuri > sys:\uri_delims.h
- Copy the file "uri_delims.h" to the directory \httpd\srclib\apr-util\uri on the build machine.
- Change directory to \httpd\srclib\apr and build APR by running "gmake -f nwgnumakefile"
- Change directory to \httpd\srclib\pcre and build DFTABLES.nlm by running "gmake -f nwgnumakefile"
- Change directory to \httpd\server and build GNECHARS.nlm by running "gmake -f nwgnumakefile"
- Copy the files "GENCHARS.nlm" and "DFTABLES.nlm" from their respective directories to the SYS: volume of a NetWare server and run them using the following commands:
    - SYS:\genchars > sys:\test_char.h
    - SYS:\dftables > sys:\chartables.c
- Copy the files "test_char.h" and "chartables.c" to the directory \httpd\os\netware on the build machine.
- Change directory to \httpd and build Apache by running "gmake -f nwgnumakefile." You can create a distribution directory by adding an install parameter to the command (ex. gmake -f nwgnumakefile install).

Additional make options

- gmake -f nwgnumakefile - Builds release versions of all of the binaries and copies them to a \release destination directory.
- gmake -f nwgnumakefile DEBUG=1 - Builds debug versions of all of the binaries and copies them to a \debug destination directory.
- gmake -f nwgnumakefile install - Creates a complete Apache distribution with binaries, docs and additional support files in a \dist\Apache2 directory.
- gmake -f nwgnumakefile installdev - Same as install but also creates a \lib and \include directory in the destination directory and copies headers and import files.
- gmake -f nwgnumakefile clean - Cleans all object files and binaries from the \release or \debug build areas depending on whether DEBUG has been defined.
- gmake -f nwgnumakefile clobber_all - Same as clean and also deletes the distribution directory if it exists.

Apache HTTP Server Version 2.0

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 nettune 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 2.0

Warning: This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

Overview of the Apache EBCDIC Port

Version 1.3 of the Apache HTTP Server is the first version which includes a port to a (non-ASCII) 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. This mainframe OS nowadays features a SVR4-derived POSIX subsystem).

The port was started initially to

- prove the feasibility of porting the Apache HTTP server to this platform
- find a "worthy and capable" successor for the venerable CERN-3.0 daemon (which was ported a couple of years ago), and to
- prove that Apache's preforking process model can on this platform easily outperform the accept-fork-serve model used by CERN by a factor of 5 or more.

This document serves as a rationale to describe some of the design decisions of the port to this machine.

Design Goals

One objective of the EBCDIC port was to maintain enough backwards compatibility with the (EBCDIC) CERN server to make the transition to the new server attractive and easy. This required the addition of a configurable method to define whether a HTML document was stored in ASCII (the only format accepted by the old server) or in EBCDIC (the native document format in the POSIX subsystem, and therefore the only realistic format in which the other POSIX tools like grep or sed could operate on the documents). The current solution to this is a "pseudo-MIME-format" which is intercepted and interpreted by the Apache server (see below). Future versions might solve the problem by defining an "ebcdic-handler" for all documents which must be converted.

Technical Solution

Since all Apache input and output is based upon the BUFF data type and its methods, the easiest solution was to add the conversion to the BUFF handling routines. The conversion must be settable at any time, so a BUFF flag was added which defines whether a BUFF object has currently enabled conversion or not. This flag is modified at several points in the HTTP protocol:

- **set** before a request is received (because the request and the request header lines are always in ASCII format)
- **set/unset** 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** before a reply header is sent (because the response header lines are always in ASCII format)
- **set/unset** 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)

Porting Notes

1. The relevant changes in the source are #ifdef'ed 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`

   Code which is needed for the SIEMENS BS2000/OSD mainframe platform only. This deals with include file differences and socket implementation topics which are only required on the BS2000/OSD platform.

2.

3. 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)

4.

5. 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. The hard coded ASCII escapes \012 and \015 which are ubiquitous in the server code are an exception: they are already the binary encoding of the ASCII \n and \r and must not be converted to ASCII a second time. This exception is only relevant for server-generated strings; and *external* EBCDIC documents are not expected to contain ASCII newline characters.

6.

7. 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 a conversion flag 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.

8.

9. For Text documents (MIME types text/plain, text/html *etc.*), an implicit translation to ASCII can be used, or (if the users prefer to store some documents in raw ASCII form for faster serving, or because the files reside on a NFS-mounted directory tree) can be served without conversion.
   Example:

   to serve files with the suffix .ahtml as a raw ASCII text/html document without implicit conversion (and suffix .ascii as ASCII text/plain), use the directives:

   ```
       AddType  text/x-ascii-html  .ahtml
       AddType  text/x-ascii-plain .ascii
   ```

   Similarly, any text/foo MIME type can be served as "raw ASCII" by configuring a MIME type "text/x-ascii-foo" for it using AddType.

10.

11. Non-text documents are always served "binary" without conversion. This seems to be the most sensible choice for, *.e.g.*, GIF/ZIP/AU file types. This of course requires the user to copy them to the mainframe host using the "rcp -b" binary switch.

12.

13. Server parsed files are always assumed to be in native (*i.e.*, EBCDIC) format as used on the machine, and are converted after processing.

14.

15. 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. An example for the latter case is the wwwcount program which we ported as well.

16.

Document Storage Notes

Binary Files

All files with a Content-Type: which does not start with text/ are regarded as *binary files* by the server and are not subject to any conversion. Examples for binary files are GIF images, gzip-compressed files and the like.

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 it from ASCII before processing.

Apache Modules' Status

Module	Status	Notes
http_core	+	
mod_access	+	
mod_actions	+	
mod_alias	+	
mod_asis	+	
mod_auth	+	
mod_auth_anon	+	
mod_auth_dbm	?	with own libdb.a
mod_autoindex	+	
mod_cern_meta	?	
mod_cgi	+	
mod_digest	+	
mod_dir	+	
mod_so	-	no shared libs
mod_env	+	
mod_example	-	(test bed only)
mod_expires	+	
mod_headers	+	
mod_imap	+	
mod_include	+	
mod_info	+	
mod_log_agent	+	
mod_log_config	+	
mod_log_referer	+	
mod_mime	+	
mod_mime_magic	?	not ported yet
mod_negotiation	+	
mod_proxy	+	
mod_rewrite	+	untested

mod_setenvif	+	
mod_speling	+	
mod_status	+	
mod_unique_id	+	
mod_userdir	+	
mod_usertrack	?	untested

Third Party Modules' Status

Module	Status	Notes
mod_jserv	-	JAVA still being ported.
mod_php3	+	mod_php3 runs fine, with LDAP and GD and FreeType libraries
mod_put	?	untested
mod_session	-	untested

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

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 2.0

Apache HTTP Server Version 2.0

Manual Page: http

NAME

httpd - Apache hypertext transfer protocol server

SYNOPSIS

httpd [**-d** *serverroot*] [**-f** *config*] [**-C** *directive*] [**-c** *directive*] [**-D** *parameter*]

httpd [**-h**] [**-l**] [**-L**] [**-v**] [**-V**] [**-t**] [**-X**]

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.

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://httpd.apache.org/. Paths in this manual may not reflect those compiled into **httpd.**

OPTIONS

-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/apache2**.

-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). If **-D** *DUMP_VHOSTS* is also set, details of the virtual host configuration will be printed.

-v Print the version of **httpd** , and then exit.

-V Print the version and build parameters of **httpd** , and then exit.

-X Run **httpd** in debug mode. Only one worker will be started and the server will not detach from the console.

FILES

/usr/local/apache2/conf/httpd.conf
/usr/local/apache2/conf/mime.types
/usr/local/apache2/conf/magic
/usr/local/apache2/logs/error_log
/usr/local/apache2/logs/access_log
/usr/local/apache2/logs/httpd.pid

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Manual Page: ab

NAME

ab - Apache HTTP server benchmarking tool

SYNOPSIS

ab [**-k**] [**-n** *requests*] [**-t** *timelimit*] [**-c** *concurrency*] [**-p** *POST file*] [**-A** *Authentication username*:*password*] [**-P** *Proxy Authentication username*:*password*] [**-H** *Custom header*] [**-C** *Cookie name*=*value*] [**-T** *content-type*] [**-v** *verbosity*]] [**-w** *output HTML*]] [**-x** *<table> attributes*]] [**-y** *<tr> attributes*]] [**-z** *<td> attributes*] [*http*://]*hostname*[:*port*]/*path*

ab [**-V**] [**-h**]

DESCRIPTION

ab is a tool for benchmarking your Apache HyperText Transfer Protocol (HTTP) server. It is designed to give you an impression of how your current Apache installation performs. This especially shows you how many requests per second your Apache installation is capable of serving.

OPTIONS

-k Enable the HTTP KeepAlive feature, i.e., perform multiple requests within one HTTP session. Default is no KeepAlive.

-n *requests* Number of requests to perform for the benchmarking session. The default is to just perform a single request which usually leads to non-representative benchmarking results.

-t *timelimit*
Maximum number of seconds to spend for benchmarking. This implies a **-n 50000** internally. Use this to benchmark the server within a fixed total amount of time. Per default there is no timelimit.

-c *concurrency*
Number of multiple requests to perform at a time. Default is one request at a time.

-p *POST file*
File containing data to POST.

-A *Authentication username*:*password*
Supply BASIC Authentication credentials to the server. The username and password are separated by a single ':' and sent on the wire uuencoded. The string is sent regardless of whether the server needs it; (i.e., has sent an 401 authentication needed).

-p *Proxy-Authentication username*:*password*

Supply BASIC Authentication credentials to a proxy en-route. The username and password are separated by a single ':' and sent on the wire uuencoded. The string is sent regardless of whether the proxy needs it; (i.e., has sent an 407 proxy authentication needed).

-C *Cookie name=value*
: Add a 'Cookie:' line to the request. The argument is typically in the form of a 'name=value' pair. This field is repeatable.

-p *Header string*
: Append extra headers to the request. The argument is typically in the form of a valid header line, containing a colon-separated field-value pair. (i.e., 'Accept-Encoding: zip/zop;8bit').

-T *content-type*
: Content-type header to use for POST data.

-v
: Set verbosity level - 4 and above prints information on headers, 3 and above prints response codes (404, 200, etc.), 2 and above prints warnings and info.

-w
: Print out results in HTML tables. Default table is two columns wide, with a white background.

-x *attributes*
: String to use as attributes for <table>. Attributes are inserted <table **here** >

-y *attributes*
: String to use as attributes for <tr>.

-z *attributes*
: String to use as attributes for <td>.

-V
: Display version number and exit.

-h
: Display usage information.

BUGS

There are various statically declared buffers of fixed length. Combined with the lazy parsing of the command line arguments, the response headers from the server and other external inputs, this might bite you.

It does not implement HTTP/1.x fully; only accepts some 'expected' forms of responses. The rather heavy use of **strstr(3)** shows up top in profile, which might indicate a performance problem; i.e., you would measure the **ab** performance rather than the server's.

SEE ALSO

httpd(8)

Apache HTTP Server Version 2.0

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://httpd.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. *On certain platforms that do not allow USR1 to be used for a graceful restart, an alternative signal may be used (such as WINCH). apachectl graceful will send the right signal for your platform.*

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 2.0

Apache HTTP Server Version 2.0

Manual Page: apxs

NAME

apxs - APache eXtenSion tool

SYNOPSIS

apxs -g [**-S** *name=value*] **-n** *modname*

apxs -q [**-S** *name=value*] *query* ...

apxs -c [**-S** *name=value*] [**-o** *dsofile*] [**-I** *incdir*] [**-D** *name=value*] [**-L** *libdir*] [**-l** *libname*] [**-Wc,***compiler-flags*] [**-Wl,***linker-flags*] *files* ...

apxs -i [**-S** *name=value*] [**-n** *modname*] [**-a**] [**-A**] *dso-file* ...

apxs -e [**-S** *name=value*] [**-n** *modname*] [**-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/modules/mod_foo.so
  chmod 755 /path/to/apache/modules/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 used the C source files for compilation while just using 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 dynamically loaded shared object. For instance with GCC you always just have to use **-fpic**. For other C compilers consult its manual page or at watch for the flags **apxs** uses to compile the object files.

For more details about DSO support in Apache read the documentation of **mod_so** or perhaps even read the **src/modules/standard/mod_so.c** source file.

OPTIONS

Common options:

-n *modname* 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 strings:

```
  CC              TARGET
  CFLAGS          SBINDIR
  CFLAGS_SHLIB    INCLUDEDIR
  LD_SHLIB        LIBEXECDIR
  LDFLAGS_SHLIB   SYSCONFDIR
  LIBS_SHLIB
```

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 *name*=*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 build 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 dynamically shared object 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 dynamically shared object. If not specified and the name cannot be guessed from the *files* list, the fallback name **mod_unknown.so** is used.

-D *name*=*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 dynamically shared objects into the server's *modules* 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 shared object 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 shared object. To simplify this step **apxs** provides an automatic way to install the shared object in its "modules" 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/modules/mod_foo.so
  chmod 755 /path/to/apache/modules/mod_foo.so
  [activating module `foo' in /path/to/apache/etc/httpd.conf]
  $ _
```

This way a line named

```
  LoadModule foo_module modules/mod_foo.so
```

is added to the configuration file if still not present. If you want to have this disabled per default 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 shared object 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/modules/mod_foo.so
  chmod 755 /path/to/apache/modules/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:

```
  $ 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
  $ _
```

because **apxs** automatically recognized C source files and object files. 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 2.0

Apache HTTP Server Version 2.0

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 web server 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 it's 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_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 2.0

Apache HTTP Server Version 2.0

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 given, 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 is this file, an entry is added. If it does exist, the password is changed.

SEE ALSO

httpd(8)

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Manual Page: htpasswd

NAME

htpasswd - Create and update user authentication files

SYNOPSIS

htpasswd [**-c**] [**-m**] *passwdfile username*
htpasswd -b [**-c**] [**-m** | **-d** | **-p** | **-s**] *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/>.

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 MD5 encryption for passwords. On Windows and TPF, this 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. Facilitates 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/doe/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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Manual Page: rotatelogs

NAME

rotatelogs - rotate Apache logs without having to kill the server

SYNOPSIS

rotatelogs *logfile rotationtime* [*offset*]

DESCRIPTION

rotatelogs is a simple program for use in conjunction with Apache's piped logfile feature, which can be used like this:

```
CustomLog "|bin/rotatelogs /var/logs/logfile 86400" common
```

This creates the files /var/logs/logfile.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. If **logfile** includes any Otherwise, the suffix .nnnnnnnnnn is automatically added and is the time in seconds. Both formats compute the start time from the beginning of the current period.

rotationtime
The rotation time in seconds.

offset
The number of minutes offset from UTC. If omitted, zero is assumed and UTC is used. For example, to use local time in the zone UTC -5 hours, specify a value of -*300* for this argument.

PORTABILITY

The following logfile format string substitutions should be supported by all *strftime*(*3*) implementations, see the *strftime*(*3*) man page for library-specific extensions.

%A	full weekday name (localized)
%a	3-character weekday name (localized)
%B	full month name (localized)
%b	3-character month name (localized)
%c	date and time (localized)
%d	2-digit day of month
%H	2-digit hour (24 hour clock)

%I	2-digit hour (12 hour clock)
%j	3-digit day of year
%M	2-digit minute
%m	2-digit month
%p	am/pm of 12 hour clock (localized)
%S	2-digit second
%U	2-digit week of year (Sunday first day of week)
%W	2-digit week of year (Monday first day of week)
%w	1-digit weekday (Sunday first day of week)
%X	time (localized)
%x	date (localized)
%Y	4-digit year
%y	2-digit year
%Z	time zone name
%%	literal `%'

SEE ALSO

httpd(8)

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Manual Page: suexec

NAME
```
     suexec - Switch User For Exec
```

SYNOPSIS
```
     No synopsis for usage, because this program is  used  inter-
     nally by Apache only.
```

DESCRIPTION
```
     suexec is the  "wrapper"  support  program  for  the  suEXEC
     behaviour for Apache.  It is run from within Apache automat-
     ically 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://httpd.apache.org/docs-2.0/suexec.html .
```

SEE ALSO
```
     httpd(8)
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

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 2.0

Apache HTTP Server Version 2.0

Apache Miscellaneous Documentation

Below is a list of additional documentation pages that apply to the Apache web server development project.

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.

Known Client Problems

> A list of problems in HTTP clients which can be mitigated by Apache.

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.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Using XSSI and ErrorDocument to configure customized international server error responses

Index

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.

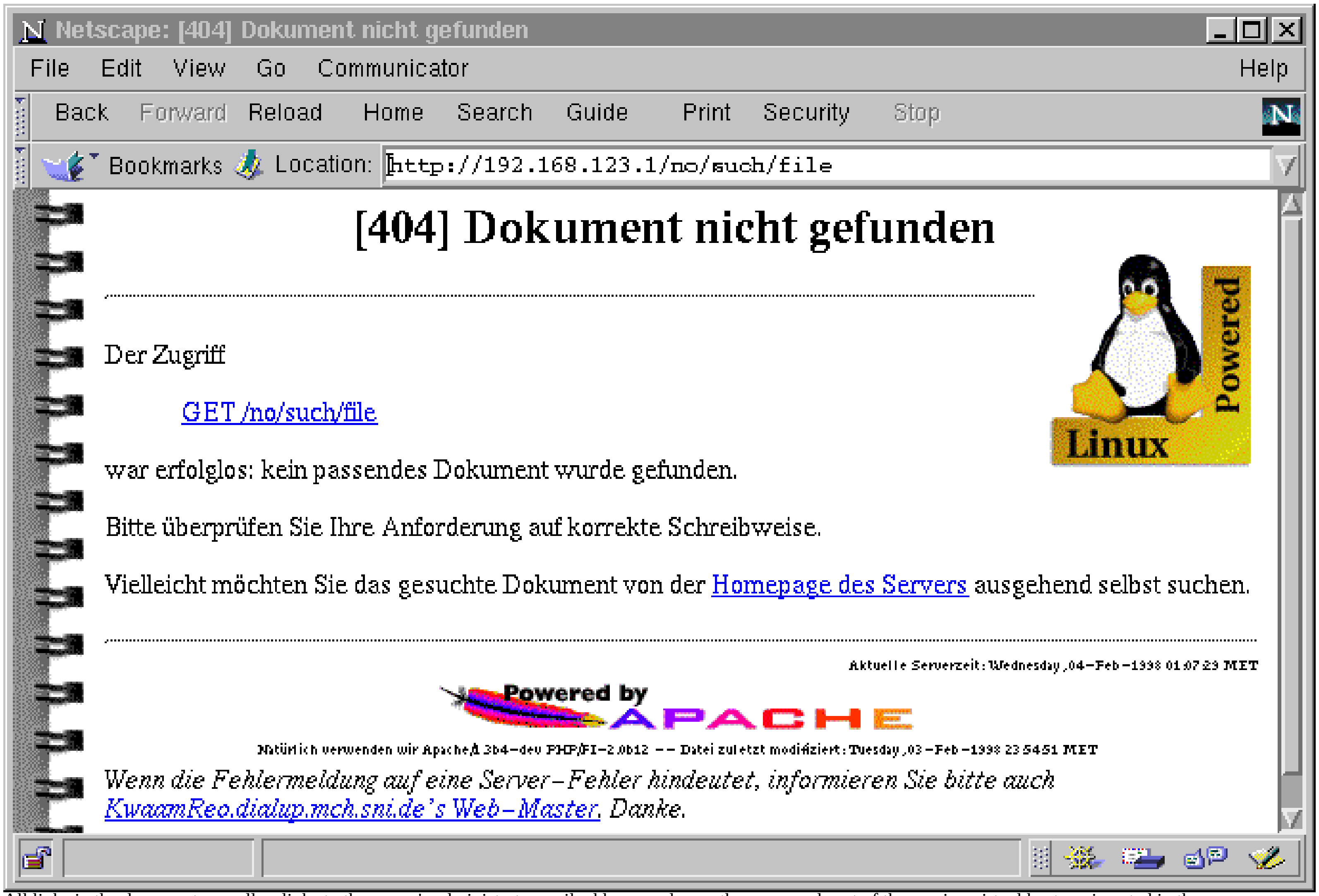
By using XSSI, all customized messages can share a homogenous and consistent style and layout, and maintenance work (changing images, changing links) is kept to a minimum because all layout information can be kept in a single file.
Error documents can be shared across different servers, or even hosts, 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 hypothetic 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:


Netscape: [404] Dokument nicht gefunden
File Edit View Go Communicator Help
Back Forward Reload Home Search Guide Print Security Stop
Bookmarks Location: http://192.168.123.1/no/such/file
Powered Linux
Powered by APACHE

[404] Dokument nicht gefunden

Der Zugriff

GET /no/such/file

war erfolglos: kein passendes Dokument wurde gefunden.

Bitte überprüfen Sie Ihre Anforderung auf korrekte Schreibweise.

Vielleicht möchten Sie das gesuchte Dokument von der Homepage des Servers ausgehend selbst suchen.

Aktuelle Serverzeit: Wednesday, 04-Feb-1998 01:07:23 MET

Natürlich verwenden wir Apache/1.3b4-dev PHP/FI-2.0b12 -- Datei zuletzt modifiziert: Tuesday, 03-Feb-1998 23:54:51 MET

Wenn die Fehlermeldung auf eine Server-Fehler hindeutet, informieren Sie bitte auch KwaamReo.dialup.mch.sni.de's Web-Master. Danke.

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
   <FilesMatch "\.shtml[.$]">
    SetOutputFilter INCLUDES
   </FilesMatch>
  </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 2.0

Apache HTTP Server Version 2.0

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 sent 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 `nettune` 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 as 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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

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/fixed.

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 behaviour, 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 2.0

Apache HTTP Server Version 2.0

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 2.0

Apache HTTP Server Version 2.0

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 Version 2.0

Apache HTTP Server Version 2.0

Module Index

Below is a list of all of the modules that come as part of the Apache distribution. See also the complete alphabetical list of all Apache directives.

Core Features and Multi-Processing Modules

core

Core Apache HTTP Server features that are always available

mpm_common

A collection of directives that are implemented by more than one multi-processing module (MPM)

mpm_netware

Multi-Processing Module implementing an exclusively threaded web server optimized for Novell NetWare

mpm_winnt

This Multi-Processing Module is optimized for Windows NT.

perchild

Multi-Processing Module allowing for daemon processes serving requests to be assigned a variety of different userids

prefork

Implements a non-threaded, pre-forking web server

worker

Multi-Processing Module implementing a hybrid multi-threaded multi-process web server

Other Modules

mod_access

Provides access control based on client hostname, IP address, or other characteristics of the client request.

mod_actions

This module provides for executing CGI scripts based on media type or request method.

mod_alias

Provides for mapping different parts of the host filesystem in the document tree and for URL redirection

mod_asis

Sends files that contain their own HTTP headers

mod_auth

User authentication using text files

mod_auth_anon

Allows "anonymous" user access to authenticated areas

mod_auth_dbm

Provides for user authentication using DBM files

mod_auth_digest
: User authentication using MD5 Digest Authentication.

mod_autoindex
: Generates directory indexes, automatically, similar to the Unix *ls* command or the Win32 *dir* shell command

mod_cache
: Content cache keyed to URIs

mod_cern_meta
: CERN httpd metafile semantics

mod_cgi
: Execution of CGI scripts

mod_cgid
: Execution of CGI scripts using an external CGI daemon

mod_charset_lite
: Specify character set translation or recoding

mod_dav
: Distributed Authoring and Versioning (WebDAV) functionality

mod_deflate
: Compress content before it is delivered to the client

mod_dir
: Provides for "trailing slash" redirects and serving directory index files

mod_env
: Modifies the environment which is passed to CGI scripts and SSI pages

mod_example
: Illustrates the Apache module API

mod_expires
: Generation of `Expires` HTTP headers according to user-specified criteria

mod_ext_filter
: Pass the response body through an external program before delivery to the client

mod_file_cache
: Caches a static list of files in memory

mod_headers
: Customization of HTTP request and response headers

mod_imap
: Server-side imagemap processing

mod_include
: Server-parsed html documents (Server Side Includes)

mod_info
: Provides a comprehensive overview of the server configuration

mod_isapi
: ISAPI Extensions within Apache for Windows

mod_log_config
: Logging of the requests made to the server

mod_mime
: Associates the request filename's extensions (e.g. .html) with the file's behavior (handlers and filters) and content (mime-type, language, character set and encoding)

mod_mime_magic

Determines the MIME type of a file by looking at a few bytes of its contents

mod_negotiation

Provides for content negotiation

mod_proxy

HTTP/1.1 proxy/gateway server

mod_rewrite

Provides a rule-based rewriting engine to rewrite requested URLs on the fly

mod_setenvif

Allows the setting of environment variables based on characteristics of the request

mod_so

This module provides for loading of executable code and modules into the server at start-up or restart time.

mod_speling

Attempts to correct mistaken URLs that users might have entered by ignoring capitalization and by allowing up to one misspelling

mod_ssl

Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols

mod_status

Provides information on server activity and performance

mod_suexec

Allows CGI scripts to run as a specified user and Group

mod_unique_id

Provides an environment variable with a unique identifier for each request

mod_userdir

Provides for user-specific directories

mod_usertrack

This module uses cookies to provide for a *clickstream* log of user activity on a site.

mod_vhost_alias

Provides for dynamically configured mass virtual hosting

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

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.

Core and Mutli-Processing Modules

Core

Core Apache features.

worker

Multi-Processing Module with Threading via Pthreads; Variable number of processes, constant number of threads/child

mpm_winnt

Multi-Processing Module with a single control process and a single server process with multiple threads for Windows NT

perchild

Multi-Processing Module with the ability to server different virtual hosts under different userids.

prefork

Non-threaded preforking processes model similar to Apache 1.3

mpm_netware

Thread only model optimized for Novell NetWare

Environment Creation

mod_env

Passing of environments to CGI scripts

mod_setenvif

Set environment variables based on client information

mod_unique_id

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.

mod_charset_lite

Configuring character set translation.

URL Mapping

mod_alias
: Mapping different parts of the host filesystem in the document tree, and URL redirection.

mod_rewrite
: Powerful URI-to-filename mapping using regular expressions

mod_userdir
: User home directories.

mod_speling
: Automatically correct minor typos in URLs

mod_vhost_alias
: 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_anon
: Anonymous user access to authenticated areas.

mod_auth_digest
: MD5 authentication

mod_auth_ldap
: User authentication using LDAP.

HTTP Response

mod_headers
: Add arbitrary HTTP headers to resources

mod_cern_meta
: Support for HTTP header metafiles.

mod_expires
: 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_cgid

Invoking CGI scripts using an external daemon.

mod_actions

Executing CGI scripts based on media type or request method.

mod_isapi

Windows ISAPI Extension support

mod_ext_filter

Filtering content with external programs.

mod_suexec

Run CGI requests as a specified user and group.

Internal Content Handlers

mod_status

Server status display

mod_info

Server configuration information

Logging

mod_log_config

User-configurable logging replacement for mod_log_common.

mod_usertrack

User tracking using Cookies

Miscellaneous

mod_imap

The imagemap file handler.

mod_proxy

Caching proxy abilities

mod_so

Support for loading modules at runtime

mod_file_cache

Caching files in memory for faster serving.

mod_dav

Class 1,2 WebDAV HTTP extensions

mod_deflate

Compression of content

mod_ssl

strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols

mod_ldap

LDAP connection pool and shared memory cache.

Development

mod_example

Demonstrates Apache API

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Directive Index

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.

- AcceptPathInfo
- AccessFileName
- Action
- AddAlt
- AddAltByEncoding
- AddAltByType
- AddCharset
- AddDefaultCharset
- AddDescription
- AddEncoding
- AddHandler
- AddIcon
- AddIconByEncoding
- AddIconByType
- AddInputFilter
- AddLanguage
- AddModuleInfo
- AddOutputFilter
- AddType
- Alias
- AliasMatch
- Allow
- AllowCONNECT
- AllowOverride
- Anonymous
- Anonymous_Authoritative
- Anonymous_LogEmail
- Anonymous_MustGiveEmail
- Anonymous_NoUserID
- Anonymous_VerifyEmail
- AuthAuthoritative
- AuthDBMAuthoritative

- AuthDBMGroupFile
- AuthDBMType
- AuthDBMUserFile
- AuthDigestAlgorithm
- AuthDigestDomain
- AuthDigestFile
- AuthDigestGroupFile
- AuthDigestNcCheck
- AuthDigestNonceFormat
- AuthDigestNonceLifetime
- AuthDigestQop
- AuthGroupFile
- AuthName
- AuthType
- AuthUserFile
- BrowserMatch
- BrowserMatchNoCase
- CacheDefaultExpire
- CacheDisable
- CacheEnable
- CacheFile
- CacheIgnoreCacheControl
- CacheIgnoreNoLastMod
- CacheLastModifiedFactor
- CacheMaxExpire
- CacheNegotiatedDocs
- CacheOn
- CharsetDefault
- CharsetOptions
- CharsetSourceEnc
- CheckSpelling
- ContentDigest
- CookieDomain
- CookieExpires
- CookieLog
- CookieName
- CookieStyle
- CookieTracking
- CoreDumpDirectory
- CustomLog
- Dav
- DavDepthInfinity

- DavLockDB
- DavMinTimeout
- DefaultIcon
- DefaultLanguage
- DefaultType
- DeflateFilterNote
- DeflateMemLevel
- DeflateWindowSize
- Deny
- Directory
- DirectoryIndex
- DirectoryMatch
- DocumentRoot
- ErrorDocument
- ErrorLog
- Example
- ExpiresActive
- ExpiresByType
- ExpiresDefault
- ExtFilterDefine
- ExtFilterOptions
- ExtendedStatus
- FileETag
- Files
- FilesMatch
- ForceLangaugePriority
- ForceType
- Group
- Header
- HeaderName
- HostnameLookups
- ISAPIAppendLogToErrors
- ISAPIAppendLogToQuery
- ISAPIFileChache
- ISAPILogNotSupported
- ISAPIReadAheadBuffer
- IdentityCheck
- IfDefine
- IfModule
- ImapBase
- ImapDefault
- ImapMenu

- Include
- IndexIgnore
- IndexOptions
- IndexOrderDefault
- KeepAlive
- KeepAliveTimeout
- LanguagePriority
- Limit
- LimitExcept
- LimitRequestBody
- LimitRequestFieldSize
- LimitRequestFields
- LimitRequestLine
- LimitXMLRequestBody
- Listen
- ListenBackLog
- LoadFile
- LoadModule
- Location
- LocationMatch
- LockFile
- LogFormat
- LogLevel
- MMapFile
- MaxClients
- MaxKeepAliveRequests
- MaxRequestsPerChild
- MaxSpareThreads
- MaxThreadsPerChild
- MetaDir
- MetaFiles
- MetaSuffix
- MimeMagicFile
- MinSpareThreads
- MultiviewsMatch
- NameVirtualHost
- NoProxy
- NumServers
- Options
- Order
- PassEnv
- PidFile

- ProxyBlock
- ProxyDomain
- ProxyErrorOverride
- ProxyMaxForwards
- ProxyPass
- ProxyPassReverse
- ProxyPreserveHost
- ProxyReceiveBufferSize
- ProxyRemote
- ProxyRequests
- ProxyTimeout
- ProxyVia
- RLimitCPU
- RLimitMEM
- RLimitNPROC
- ReadmeName
- Redirect
- RedirectMatch
- RedirectPermanent
- RedirectTemp
- RemoveCharset
- RemoveEncoding
- RemoveHandler
- RemoveInputFilter
- RemoveLanguage
- RemoveOutputFilter
- RemoveType
- RequestHeader
- Require
- RewriteBase
- RewriteCond
- RewriteEngine
- RewriteLock
- RewriteLog
- RewriteLogLevel
- RewriteMap
- RewriteOptions
- RewriteRule
- SSIEndTag
- SSIErrorMsg
- SSIStartTag
- SSITimeFormat

- SSIUndefinedEcho
- SSLCACertificateFile
- SSLCACertificatePath
- SSLCARevocationFile
- SSLCARevocationPath
- SSLCertificateChainFile
- SSLCertificateFile
- SSLCertificateKeyFile
- SSLCipherSuite
- SSLEngine
- SSLLog
- SSLLogLevel
- SSLMutex
- SSLOptions
- SSLPassPhraseDialog
- SSLProtocol
- SSLRandomSeed
- SSLRequire
- SSLRequireSSL
- SSLSessionCache
- SSLSessionCacheTimeout
- SSLVerifyClient
- SSLVerifyDepth
- Satisfy
- ScoreBoardFile
- Script
- ScriptAlias
- ScriptAliasMatch
- ScriptInterpreterSource
- ScriptLog
- ScriptLogBuffer
- ScriptLogLength
- SendBufferSize
- ServerAdmin
- ServerAlias
- ServerLimit
- ServerName
- ServerPath
- ServerRoot
- ServerSignature
- ServerTokens
- SetEnv

- SetEnvIf
- SetEnvIfNoCase
- SetHandler
- SetInputFilter
- SetOutputFilter
- StartServers
- StartThreads
- SuexecUserGroup
- ThreadLimit
- ThreadsPerChild
- TimeOut
- TransferLog
- TypesConfig
- UnsetEnv
- UseCanonicalName
- User
- UserDir
- VirtualDocumentRoot
- VirtualDocumentRootIP
- VirtualHost
- VirtualScriptAlias
- VirtualScriptAliasIP
- XBitHack

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

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:

MPM

A module with status "MPM" is a Multi-Processing Module. Unlike the other types of modules, Apache must have one and only one MPM in use at any time. This type of module is responsible for basic request handling and dispatching.

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 for, 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 2 distribution, the version in which it was introduced should be listed here.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

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*
Deprecated: *see other*

Each of the directive's attributes, complete with possible values where possible, are described in this document.

Directive Terms

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 *extension*s with or without the leading dot. In addition, *extension*s 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, srm.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.

MPM

A directive labeled as having "MPM" status is provided by a Multi-Processing Module. This type of directive will be available if and only if you are using one of the MPMs listed on the Module line of the directive definition.

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.*"

Deprecated

If this directive is eliminated since the Apache version 1 distribution, the directive or option that replaces the behavior should be cited here. In general, directives, features, and options are only deprecated to minimize debugging of conflicting features, or if the feature can only continue to be supported in an alternate manner.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module core

Description:	Core Apache HTTP Server features that are always available
Status:	Core

Directives

- AcceptPathInfo
- AccessFileName
- AddDefaultCharset
- AllowOverride
- AuthName
- AuthType
- ContentDigest
- DefaultType
- Directory
- DirectoryMatch
- DocumentRoot
- ErrorDocument
- ErrorLog
- FileETag
- Files
- FilesMatch
- ForceType
- HostnameLookups
- IdentityCheck
- IfDefine
- IfModule
- Include
- KeepAlive
- KeepAliveTimeout
- Limit
- LimitExcept
- LimitRequestBody
- LimitRequestFieldSize
- LimitRequestFields

AcceptPathInfo Directive

Description:	Controls whether requests can contain trailing pathname information
Syntax:	AcceptPathInfo On|Off|Default
Default:	`AcceptPathInfo Default`
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core
Compatibility:	Available in Apache 2.0.30 and later

This directive controls whether requests that contain trailing pathname information that follows an actual filename (or non-existent file in an existing directory) will be accepted or rejected. The trailing pathname information can be made available to scripts in the PATH_INFO environment variable.

For example, assume the location `/test/` points to a directory that contains only the single file `here.html`. Then requests for `/test/here.html/more` and `/test/nothere.html/more` both collect `/more` as PATH_INFO.

The three possible arguments for the `AcceptPathInfo` directive are:

`off`

A request will only be accepted if it maps to a literal path that exists. Therefore a request with trailing pathname information after the true filename such as `/test/here.html/more` in the above example will return a 404 NOT FOUND error.

`on`

A request will be accepted if a leading path component maps to a file that exists. The above example `/test/here.html/more` will be accepted if `/test/here.html` maps to a valid file.

`default`

The treatment of requests with trailing pathname information is determined by the handler responsible for the request. The core handler for normal files defaults to rejecting PATH_INFO. Handlers that serve scripts, such as cgi-script and isapi-isa, generally accept PATH_INFO by default.

The primary purpose of the `AcceptPathInfo` directive is to allow you to override the handler's choice of accepting or rejecting PATH_INFO. This override is required, for example, when you use a filter, such as INCLUDES, to generate content based on PATH_INFO. The core handler would usually reject the request, so you can use the following configuration to enable such a script:

```
<Files "mypaths.shtml">
Options +Includes
SetOutputFilter INCLUDES
AcceptPathInfo on
</Files>
```

AccessFileName Directive

Description:	Sets the name of the .htaccess file
Syntax:	AccessFileName *filename* [*filename*] ...
Default:	`AccessFileName .htaccess`
Context:	server config, virtual host
Status:	Core
Module:	core

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`
- Configuration Files

AddDefaultCharset Directive

Description:	Specifies the default character set to be added for a response without an explicit character set
Syntax:	AddDefaultCharset On|Off|*charset*
Default:	`AddDefaultCharset Off`
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

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. For example:

```
AddDefaultCharset utf-8
```

AllowOverride Directive

Description:	Sets the types of directives that are allowed in .htaccess files
Syntax:	AllowOverride All|None|*directive-type* [*directive-type*] ...
Default:	`AllowOverride All`
Context:	directory
Status:	Core
Module:	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 (`DefaultType`, `ErrorDocument`, `ForceType`, `LanguagePriority`, `SetHandler`, `SetInputFilter`, `SetOutputFilter`, and `mod_mime` Add* and Remove* directives, *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`).

Example:

```
AllowOverride AuthConfig Indexes
```

See also

- AccessFileName
- Configuration Files

AuthName Directive

Description:	Sets the authorization realm for use in HTTP authentication
Syntax:	AuthName *auth-domain*
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Core
Module:	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.

For example:

```
AuthName "Top Secret"
```

The string provided for the `AuthRealm` is what will appear in the password dialog provided by most browsers.

See also

- Authentication, Authorization, and Access Control

AuthType Directive

Description:	Selects the type of user authentication
Syntax:	AuthType Basic|Digest
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Core
Module:	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.

See also

- Authentication, Authorization, and Access Control

ContentDigest Directive

Description:	Enables the generation of Content-MD5 HTTP Response headers
Syntax:	ContentDigest on|off
Default:	`ContentDigest off`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Core
Module:	core
Compatibility:	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.

DefaultType Directive

Description:	Sets the MIME content-type that will be sent if the server cannot determine a type in any other way
Syntax:	DefaultType *MIME-type*
Default:	`DefaultType text/html`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	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.

Note that unlike `ForceType`, this directive is only provides the default mime-type. All other mime-type definitions, including filename extensions, that might identify the media type will override this default.

<Directory> Directive

Description:	Enclose a group of directives that apply only to the named file-system directory and sub-directories
Syntax:	<Directory *directory-path*> ... </Directory>
Context:	server config, virtual host
Status:	Core
Module:	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. 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>
```

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 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>
```

The regular expression section 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> Directive

Description:	Enclose a group of directives that apply only to file-system directories that match a regular expression and their subdirectories
Syntax:	<Directory *regex*> ... </Directory>
Context:	server config, virtual host
Status:	Core
Module:	core

`<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
- How Directory, Location and Files sections work for an explanation of how these different sections are combined when a request is received

DocumentRoot Directive

Description:	Sets the directory that forms the main document tree visible from the web
Syntax:	DocumentRoot *directory-path*
Default:	`DocumentRoot /usr/local/apache/htdocs`
Context:	server config, virtual host
Status:	Core
Module:	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`.

The `DocumentRoot` should be specified without a trailing slash.

See also

- Mapping URLs to Filesystem Location

ErrorDocument Directive

Description:	Specifies what the server will return to the client in case of an error
Syntax:	ErrorDocument *error-code document*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Quoting syntax for text messages is different in Apache 2.0

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 URL or a message. 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. Alternatively, a message can be provided to be displayed by the browser. 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.**

Prior to version 2.0, messages were indicated by prefixing them with a single unmatched double quote character.

See also

- documentation of customizable responses

ErrorLog Directive

Description:	Sets the name of the file to which the server will log errors
Syntax:	ErrorLog *file-path*|syslog[:*facility*]
Default:	`ErrorLog logs/error_log (Unix) ErrorLog logs/error.log (Windows and OS/2)`
Context:	server config, virtual host
Status:	Core
Module:	core

The `ErrorLog` 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.

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`
- Apache Log Files

FileETag Directive

Description:	Configures the file attributes used to create the ETag HTTP response header
Syntax:	FileETag *component* ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

The `FileETag` directive configures the file attributes that are used to create the ETag (entity tag) response header field when the document is based on a file. (The ETag value is used in cache management to save network bandwidth.) In Apache 1.3.22 and earlier, the ETag value was *always* formed from the file's inode, size, and last-modified time (mtime). The FileETag directive allows you to choose which of these -- if any -- should be used. The recognized keywords are:

INode

The file's i-node number will be included in the calculation

MTime

The date and time the file was last modified will be included

Size

The number of bytes in the file will be included

All

All available fields will be used (equivalent to '`FileETag INode MTime Size`')

None

If a document is file-based, no ETag field will be included in the response

The INode, MTime, and Size keywords may be prefixed with either '+' or '-', which allow changes to be made to the default setting inherited from a broader scope. Any keyword appearing without such a prefix immediately and completely cancels the inherited setting.

If a directory's configuration includes '`FileETag INode MTime Size`', and a subdirectory's includes '`FileETag -INode`', the setting for that subdirectory (which will be inherited by any sub-subdirectories that don't override it) will be equivalent to '`FileETag MTime Size`'.

<Files> Directive

Description:	Contains that directives that apply to matched filenames
Syntax:	<Files *filename*> ... </Files>
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

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> Directive

Description:	Contains that directives that apply to regular-expression matched filenames
Syntax:	<FilesMatch *regex*> ... </FilesMatch>
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

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

ForceType Directive

Description:	Forces all matching files to be served with the specified MIME content-type
Syntax:	ForceType *mime-type*
Context:	directory, .htaccess
Status:	Core
Module:	core
Compatibility:	Moved to the core in Apache 2.0

When placed into an `.htaccess` file or a `<Directory>`, or `<Location>` or `<Files>` section, this directive forces all matching files to be served with the content type identification given by *mime-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 unlike `DefaultType`, this directive overrides all mime-type associations, including filename extensions, that might identify the media type.

HostnameLookups Directive

Description:	Enables DNS lookups on client IP addresses
Syntax:	HostnameLookups on|off|double
Default:	`HostnameLookups off`
Context:	server config, virtual host, directory
Status:	Core
Module:	core

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 is 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

Description:	Enables logging of the RFC1413 identity of the remote user
Syntax:	IdentityCheck on|off
Default:	`IdentityCheck off`
Context:	
Status:	Core
Module:	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.

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

Description:	Encloses directives that will be processed only if a test is true at startup
Syntax:	<IfDefine [!]*parameter-name*> ... </IfDefine>
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

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 `-D`*parameter-*, 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 modules/mod_rewrite.so
LoadModule proxy_module   modules/libproxy.so
</IfDefine>
```

<IfModule> Directive

Description:	Encloses directives that are processed conditional on the presence of absence of a specific module
Syntax:	<IfModule [!]*module-name*> ... </IfModule>
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

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 included in Apache -- either compiled in or dynamically loaded using `LoadModule`. The second format reverses the test, and only processes the directives if *module name* is **not** included.

The *module name* argument is 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

Description:	Includes other configuration files from within the server configuration files
Syntax:	Include *file-path*|*directory-path*
Context:	server config
Status:	Core
Module:	core

This directive allows inclusion of other configuration files from within the server configuration files.

If `Include` 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.

The file path specified may be a fully qualified path (i.e. starting with a slash), or may be relative to the `ServerRoot` directory.

Examples:

```
Include /usr/local/apache/conf/ssl.conf
Include /usr/local/apache/conf/vhosts/
```

Or, providing paths relative to your `ServerRoot` directory:

```
Include conf/ssl.conf
Include conf/vhosts/
```

Make sure that an included directory does not contain any stray files, such as editor temporary files, for example, as Apache will attempt to read them in and use the contents as configuration directives, which may cause the server to fail on start up. Running `apachectl configtest` will give you a list of the files that are being processed during the configuration check:

```

root@host# apachectl configtest
 Processing config directory: /usr/local/apache/conf/vhosts
 Processing config file: /usr/local/apache/conf/vhosts/vhost1
 Processing config file: /usr/local/apache/conf/vhosts/vhost2
Syntax OK
```

This will help in verifying that you are getting only the files that you intended as part of your configuration.

See also

- apachectl

KeepAlive Directive

Description:	Turns on or off HTTP persistent connections.
Syntax:	KeepAlive on|off
Default:	`KeepAlive On`
Context:	server config
Status:	Core
Module:	core

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.

See also

- `MaxKeepAliveRequests`

KeepAliveTimeout Directive

Description:	Sets the amount of time the server will wait for subsequent requests on a persistent connection
Syntax:	KeepAliveTimeout *seconds*
Default:	`KeepAliveTimeout 15`
Context:	server config
Status:	Core
Module:	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.

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

Description:	Restrict access controls to only certain HTTP methods
Syntax:	<Limit *method* [*method*] ... > ... </Limit>
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	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

Description:	Restrict access controls to all HTTP methods except the named ones
Syntax:	<LimitExcept *method* [*method*] ... > ... </LimitExcept>
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

`<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.

For example:

```
<LimitExcept POST GET>
Require valid-user
<LimitExcept>
```

LimitRequestBody Directive

Description:	Restricts the total size of the HTTP request body sent from the client
Syntax:	LimitRequestBody *bytes*
Default:	`LimitRequestBody 0`
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

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.

If, for example, you are permitting file upload to a particular location, and wich to limit the size of the uploaded file to 100K, you might use the following directive:

```
LimitRequestBody 102400
```

LimitRequestFieldSize Directive

Description:	Limits the size of the HTTP request header allowed from the client
Syntax:	LimitRequestFieldsize *bytes*
Default:	`LimitRequestFieldsize 8190`
Context:	server config
Status:	Core
Module:	core

This directive specifies the number of *bytes* from 0 to the value of the compile-time constant `DEFAULT_LIMIT_REQUEST_FIELDSIZE` (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.

For example:

```
LimitRequestFieldSize 16380
```

Under normal conditions, the value should not be changed from the default.

LimitRequestFields Directive

Description:	Limits the number of HTTP request header fields that will be accepted from the client
Syntax:	LimitRequestFields *number*
Default:	`LimitRequestFields 100`
Context:	server config
Status:	Core
Module:	core

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.

For example:

```
LimitRequestFields 50
```

LimitRequestLine Directive

Description:	Limit the size of the HTTP request line that will be accepted from the client
Syntax:	LimitRequestLine *bytes*
Default:	`LimitRequestLine 8190`
Context:	server config
Status:	Core
Module:	core

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.

For example:

```
LimitRequestLine 16380
```

Under normal conditions, the value should not be changed from the default.

LimitXMLRequestBody Directive

Description:	Limits the size of an XML-based request body
Syntax:	LimitXMLRequestBody *number*
Default:	`LimitXMLRequestBody 1000000`
Context:	server config
Status:	Core
Module:	core

Limit (in bytes) on maximum size of an XML-based request body. A value of `0` will disable any checking.

Example:

```
LimitXMLRequestBody 0
```

<Location> Directive

Description:	Applies the enclosed directives only to matching URLs
Syntax:	<Location URL-path|URL> ... </Location>
Context:	server config, virtual host
Status:	Core
Module:	core

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 a URL-path of the form `/path/`. No scheme, hostname, port, or query string may be included. 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.

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>
```

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> Directive

Description:	Applies the enclosed directives only to regular-expression matching URLs
Syntax:	<LocationMatch *regex*> ... </Location>
Context:	server config, virtual host
Status:	Core
Module:	core

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

LogLevel Directive

Description:	Controls the verbosity of the ErrorLog
Syntax:	LogLevel *level*
Default:	`LogLevel warn`
Context:	server config, virtual host
Status:	Core
Module:	core

`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.

For example:

```
LogLevel notice
```

MaxKeepAliveRequests Directive

Description:	Sets the number of requests allowed on a persistent connection
Syntax:	MaxKeepAliveRequests *number*
Default:	`MaxKeepAliveRequests 100`
Context:	server config
Status:	Core
Module:	core

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.

For example:

```
MaxKeepAliveRequests 500
```

NameVirtualHost Directive

Description:	Configures an IP address for name-virtual hosting
Syntax:	NameVirtualHost *addr*[:*port*]
Context:	server config
Status:	Core
Module:	core

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, *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
```

IPv6 addresses must be enclosed in square brackets, as shown in the following example:

```
NameVirtualHost [fe80::a00:20ff:fea7:ccea]:8080
```

See also

- See also: Virtual Hosts documentation

Options Directive

Description:	Configures what features are available in a particular directory
Syntax:	Options [+|-]*option* [[+|-]*option*] ...
Default:	`Options All`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Core
Module:	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 ScriptAliase'd directories.

Indexes

If a URL which maps to a directory is requested, and the 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`.

RLimitCPU Directive

Description:	Limits the CPU consumption of processes launched by Apache children
Syntax:	RLimitCPU *number*|max [*number*|max]
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	Moved in version 2.0 to the MPMs

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`
- `RLimitNPROC`

RLimitMEM Directive

Description:	Limits the memory consumption of processes launched by Apache children
Syntax:	RLimitMEM *number*|max [*number*|max]
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	Moved in version 2.0 to the MPMs.

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`
- `RLimitNPROC`

RLimitNPROC Directive

Description:	Limits the number of processes that can be launched by processes launched by Apache children
Syntax:	RLimitNPROC *number*|max [*number*|max]
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	Moved in version 2.0 to the MPMs.

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`
- `RLimitCPU`

Require Directive

Description:	Selects which authenticated users can access a resource
Syntax:	Require *entity-name* [*entity-name*] ...
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Core
Module:	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`
- `mod_access`

Satisfy Directive

Description:	Configures how host-level access control and user authentication interact
Syntax:	Satisfy any|all
Default:	`Satisfy all`
Context:	directory, .htaccess
Status:	Core
Module:	core

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.

For example, if you wanted to let people on your network have unrestricted access to a portion of your website, but require that people outside of your network provide a password, you could use a configuration similar to the following:

```
Require valid-user
Allow from 192.168.1
Satisfy any
```

See also

- `Allow`
- `Require`

ScriptInterpreterSource Directive

Description:	Controls how the interpreter for CGI scripts is located
Syntax:	ScriptInterpreterSource registry|script
Default:	`ScriptInterpreterSource script`
Context:	directory, .htaccess
Status:	Core
Module:	core
Compatibility:	Win32 only

This directive is used to control how Apache 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.

ServerAdmin Directive

Description:	Sets the email address that the server includes in error messages sent to the client
Syntax:	ServerAdmin *email-address*
Context:	server config, virtual host
Status:	Core
Module:	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

Description:	Sets alternate names for a host used when matching requests to name-virtual hosts
Syntax:	ServerAlias *hostname* [*hostname*] ...
Context:	virtual host
Status:	Core
Module:	core

The `ServerAlias` directive sets the alternate names for a host, for use with name-based virtual hosts.

```
<VirtualHost *>
ServerName server.domain.com
ServerAlias server server2.domain.com server2
...
</VirtualHost>
```

See also

- Apache Virtual Host documentation

ServerName Directive

Description:	Sets the hostname and port that the server uses to identify itself
Syntax:	ServerName *fully-qualified-domain-name[:port]*
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	In version 2.0, this directive supersedes the functionality of the Port directive from version 1.3.

The `ServerName` directive sets the hostname and port that the server uses to identify itself. This is used when creating redirection URLs. For example, if the name of the machine hosting the webserver is `simple.example.com`, but the machine also has the DNS alias `www.example.com` and you wish the webserver to be so identified, the following directive should be used:

```
ServerName www.example.com:80
```

If no `ServerName` is specified, then the server attempts to deduce the hostname by performing a reverse lookup on the IP address. If no port is specified in the servername, then the server will use the port from the incoming request. For optimal reliability and predictability, you should specify an explicit hostname and port using the `ServerName` directive.

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 the description of the `UseCanonicalName` directive for settings which determine whether self-referential URL's (e.g., by the `mod_dir` module) will refer to the specified port, or to the port number given in the client's request.

See also

- DNS Issues
- Apache virtual host documentation
- `UseCanonicalName`
- `NameVirtualHost`
- `ServerAlias`

ServerPath Directive

Description:	Sets the legacy URL pathname for a name-virtual host that is accessed by an incompatible browser
Syntax:	ServerPath *directory-path*
Context:	virtual host
Status:	Core
Module:	core

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

Description:	Sets the base directory for the server installation
Syntax:	ServerRoot *directory-path*
Default:	`ServerRoot /usr/local/apache`
Context:	server config
Status:	Core
Module:	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`
- the security tips for information on how to properly set permissions on the ServerRoot

ServerSignature Directive

Description:	Configures the footer on server-generated documents
Syntax:	ServerSignature On|Off|EMail
Default:	`ServerSignature Off`
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

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

Description:	Configures the Server HTTP response header
Syntax:	ServerTokens Minimal|ProductOnly|OS|Full
Default:	`ServerTokens Full`
Context:	server config
Status:	Core
Module:	core

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.

SetHandler Directive

Description:	Forces all matching files to be processed by a handler
Syntax:	SetHandler *handler-name*
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core
Compatibility:	Moved into the core in Apache 2.0

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 httpd.conf:

```
<Location /status>
SetHandler server-status
</Location>
```

SetInputFilter Directive

Description:	Sets the filters that will process client requests and POST input
Syntax:	SetInputFilter *filter*[;*filter*...]
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

The `SetInputFilter` directive sets the filter or filters which will process client requests and POST input when they are received by the server. This is in addition to any filters defined elsewhere, including the `AddInputFilter` directive.

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content.

See also

- Filters documentation

SetOutputFilter Directive

Description:	Sets the filters that will process responses from the server
Syntax:	SetOutputFilter *filter* [*filter*] ...
Context:	server config, virtual host, directory, .htaccess
Status:	Core
Module:	core

The `SetOutputFilter` directive sets the filters which will process responses from the server before they are sent to the client. This is in addition to any filters defined elsewhere, including the `AddOutputFilter` directive.

For example, the following configuration will process all files in the `/www/data/` directory for server-side includes.

```
<Directory /www/data/>
  SetOutputFilter INCLUDES
</Directory>
```

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content.

See also

- Filters documentation

TimeOut Directive

Description:	Defines the amount of time the server will wait for certain events before failing a request
Syntax:	TimeOut *number*
Default:	`TimeOut 300`
Context:	server config
Status:	Core
Module:	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

Description:	Configures how the server determines its own name and port
Syntax:	UseCanonicalName on|off|dns
Default:	`UseCanonicalName on`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Core
Module:	core

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` Apache will use the hostname and port specified in the `ServerName` directive 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`
- `Listen`

<VirtualHost> Directive

Description:	Contains directives that apply only to a specific hostname or IP address
Syntax:	<VirtualHost *addr*[:*port*] [*addr*[:*port*]] ...> ... </VirtualHost>
Context:	server config
Status:	Core
Module:	core

`<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>
```

IPv6 addresses must be specified in square brackets because the optional port number could not be determined otherwise. An IPv6 example is shown below:

```
<VirtualHost [fe80::a00:20ff:fea7:ccea]>
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 Virtual Host 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)).

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 `Listen` 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 `Listen`.

See also

- Apache Virtual Host documentation
- Warnings about DNS and Apache
- Setting which addresses and ports Apache uses
- 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 2.0

Apache HTTP Server Version 2.0

Apache Module mpm_common

Description:	A collection of directives that are implemented by more than one multi-processing module (MPM)
Status:	MPM

Directives

- CoreDumpDirectory
- Group
- Listen
- ListenBackLog
- LockFile
- MaxClients
- MaxRequestsPerChild
- MaxSpareThreads
- MaxThreadsPerChild
- MinSpareThreads
- NumServers
- PidFile
- ScoreBoardFile
- SendBufferSize
- ServerLimit
- StartServers
- StartThreads
- ThreadLimit
- ThreadsPerChild
- User

CoreDumpDirectory Directive

Description:	Sets the directory where Apache attempts to switch before dumping core
Syntax:	CoreDumpDirectory *directory*
Default:	`See usage for the default setting`
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`, `prefork`, `mpm_winnt`

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.

Group Directive

Description:	Sets the group under which the server will answer requests
Syntax:	Group *unix-group*
Default:	`Group #-1`
Context:	server config, virtual host
Status:	MPM
Module:	`worker`, `perchild`, `prefork`

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< is no longer supported. To implement the suEXEC wrapper with Apache 2.0, use the `SuexecUserGroup` directive. SECURITY: See `User` for a discussion of the security considerations.

Listen Directive

Description:	Sets the IP addresses and ports that the server listens to
Syntax:	Listen [*IP-address*:]*portnumber*
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`, `prefork`, `mpm_winnt`

The `Listen` directive instructs Apache to listen to only specific IP addresses or ports; by default it responds to requests on all IP interfaces. The Listen directive is now a required directive. If it is not in the config file, the server will fail to start. This is a change from previous versions of Apache.

The Listen directive tells the server to accept incoming requests on the specified port or address-and-port combination. If only a port number is specified, the server listens to the given port on all interfaces. 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
```

IPv6 addresses must be surrounded in square brackets, as in the following example:

```
Listen [fe80::a00:20ff:fea7:ccea]:80
```

See also

- DNS Issues
- Setting which addresses and ports Apache uses

ListenBackLog Directive

Description:	Maximum length of the queue of pending connections
Syntax:	ListenBacklog *backlog*
Default:	`ListenBacklog 511`
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`, `prefork`, `mpm_winnt`

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.

LockFile Directive

Description:	Location of the accept serialization lock file
Syntax:	LockFile *filename*
Default:	`LockFile logs/accept.lock`
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`, `prefork`

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.

MaxClients Directive

Description:	Maximum number of child processes that will be created to serve requests
Syntax:	MaxClients *number*
Default:	`>MaxClients 8 (with threads) MaxClients 256`
Context:	server config
Status:	MPM
Module:	`worker`, `prefork`

The `MaxClients` directive sets the limit on the number of child processes that will be created to serve requests. When the server is built without threading, no more than this number of clients can be served simultaneously. To configure more than 256 clients with the prefork MPM, you must use the `ServerLimit` directive. To configure more than 1024 clients with the worker MPM, you must use the `ServerLimit` and `ThreadLimit` directives.

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.

When the server is compiled with threading, then the maximum number of simultaneous requests that can be served is obtained from the value of this directive multiplied by `ThreadsPerChild`.

MaxRequestsPerChild Directive

Description:	Limit on the number of requests that an individual child server will handle during its life
Syntax:	MaxRequestsPerChild *number*
Default:	`MaxRequestsPerChild 10000`
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`, `prefork`, `mpm_winnt`

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.

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.

MaxSpareThreads Directive

Description:	Maximum number of idle threads
Syntax:	MaxSpareThreads *number*
Default:	`MaxSpareThreads 10 (Perchild) or 500 (worker)`
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`

Maximum number of idle threads. Different MPMs deal with this directive differently. `perchild` monitors the number of idle threads on a per-child basis. If there are too many idle threads in that child, the server will begin to kill threads within that child.

`worker` deals with idle threads on a server-wide basis. If there are too many idle threads in the server then child processes are killed until the number of idle threads is less than this number.

See also

- `MinSpareThreads`
- `StartServers`

MaxThreadsPerChild Directive

Description:	Maximum number of threads per child process
Syntax:	MaxThreadsPerChild *number*
Default:	`MaxThreadsPerChild 64`
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`

Maximum number of threads per child. For MPMs with a variable number of threads per child, this directive sets the maximum number of threads that will be created in each child process. To increase this value beyond its default, it is necessary to change the value of the compile-time define `HARD_THREAD_LIMIT` and recompile the server.

MinSpareThreads Directive

Description:	Minimum number of idle threads available to handle request spikes
Syntax:	MinSpareServers *number*
Default:	`MinSpareThreads 5 (Perchild) or 250 (worker)`
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`

Minimum number of idle threads to handle request spikes. Different MPMs deal with this directive differently. `perchild` monitors the number of idle threads on a per-child basis. If there aren't enough idle threads in that child, the server will begin to create new threads within that child.

`worker` deals with idle threads on a server-wide basis. If there aren't enough idle threads in the server then child processes are created until the number of idle threads is greater than number.

See also

- `MaxSpareThreads`
- `StartServers`

NumServers Directive

Description:	Total number of children alive at the same time
Syntax:	NumServers *number*
Default:	`NumServers 2`
Context:	server config
Status:	MPM
Module:	`perchild`

Number of children alive at the same time. MPMs that use this directive do not dynamically create new child processes so this number should be large enough to handle the requests for the entire site.

PidFile Directive

Description:	Sets the file where the server records the process ID of the daemon
Syntax:	PidFile *filename*
Default:	`PidFile logs/httpd.pid`
Context:	server config
Status:	MPM
Module:	`worker`, `perchilde`, `prefork`, `mpm_winnt`

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`.

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.

ScoreBoardFile Directive

Description:	Location of the file used to store coordination data for the child processes
Syntax:	ScoreBoardFile *file-path*
Default:	`ScoreBoardFile logs/apache_status`
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`, `prefork`

Apache uses a scoreboard to communicate between its parent and child processes. Some architectures require a file to facilitate this communication. If the file is left unspecified, Apache first attempts to create the scoreboard entirely in memory (using anonymous shared memory) and, failing that, will attempt to create the file on disk (using file-based shared memory). Specifying this directive causes Apache to always create the file on the disk.

File-based shared memory is useful for third-party applications that require direct access to the scoreboard.

If you 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.

See also

- Stopping and Restarting Apache

SendBufferSize Directive

Description:	TCP buffer size
Syntax:	SendBufferSize *bytes*
Context:	server config
Status:	MPM
Module:	`worker`, `perchild`, `prefork`, `mpm_winnt`

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).

ServerLimit Directive

Description:	Upper limit on configurable number of processes
Syntax:	ServerLimit *number*
Default:	`ServerLimit 256 (prefork), ServerLimit 16 (worker)`
Context:	server config
Status:	MPM
Module:	`worker`, `prefork`

For the `prefork` MPM, this directive sets the maximum configured value for `MaxClients` for the lifetime of the Apache process. For the worker MPM, this directive in combination with `ThreadLimit` sets the maximum configured value for `MaxClients` for the lifetime of the Apache process. Any attempts to change this directive during a restart will be ignored, but `MaxClients` can be modified during a restart.

Special care must be taken when using this directive. If `ServerLimit` is set to a value much higher than necessary, extra, unused shared memory will be allocated. If both `ServerLimit` and `MaxClients` are set to values higher than the system can handle, Apache may not start or the system may become unstable.

With the `prefork` MPM, use this directive only if you need to set `MaxClients` higher higher than 256. Do not set the value of this directive any higher than what you might want to set `MaxClients` to.

With the `worker` MPM, use this directive only if your `MaxClients` and `ThreadsPerChild` settings require more than 16 server processes. Do not set the value of this directive any higher than the number of server processes required by what you may want for `MaxClients` and `ThreadsPerChild`.

StartServers Directive

Description:	Number of child server processes created at startup
Syntax:	StartServers *number*
Default:	`StartServers 5`
Context:	server config
Status:	MPM
Module:	`worker`

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.

See also

- `MinSpareThreads`
- `MaxSpareThreads`

StartThreads Directive

Description:	Nubmer of threads each child creates on startup
Syntax:	StartThreads *number*
Default:	`StartThreads 5`
Context:	server config
Status:	MPM
Module:	`perchild`

Number of threads each child creates on startup. As the number of threads is dynamically controlled depending on the load, there is usually little reason to adjust this parameter.

ThreadLimit Directive

Description:	Sets the upper limit on the configurable number of threads per child process
Syntax:	ThreadLimit *number*
Default:	`ThreadLimit 64`
Context:	server config
Status:	MPM
Module:	`worker`

This directive sets the maximum configured value for `ThreadsPerChild` for the lifetime of the Apache process. Any attempts to change this directive during a restart will be ignored, but `ThreadsPerChild` can be modified during a restart up to the value of this directive.

Special care must be taken when using this directive. If `ThreadLimit` is set to a value much higher than

`ThreadsPerChild`, extra unused shared memory will be allocated. If both `ThreadLimit` and `ThreadsPerChild` are set to values higher than the system can handle, Apache may not start or the system may become unstable.

Use this directive only if you need to set `ThreadsPerChild` higher than 64. Do not set the value of this directive any higher than what you might want to set `ThreadsPerChild` to.

ThreadsPerChild Directive

Description:	Number of threads created by each child process
Syntax:	ThreadsPerChild *number*
Default:	`ThreadsPerChild 50`
Context:	server config
Status:	MPM
Module:	`worker`, `mpm_winnt`

This directive sets the number of threads created by each child process. The child creates these threads at startup and never creates more. if using an MPM like mpmt_winnt, where there is only one child process, this number should be high enough to handle the entire load of the server. If using an MPM like worker, where there are multiple child processes, the total number of threads should be high enough to handle the common load on the server.

User Directive

Description:	The userid under which the server will answer requests
Syntax:	User *unix-userid*
Default:	`User #-1`
Context:	server config, virtual host
Status:	MPM
Module:	`worker`, `perchild`, `prefork`

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 `CacheRoot`).

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>` is no longer supported. To configure your server for suexec use `SuexecUserGroup`.

Security

Don't set `User` (or `Group`) to `root` unless you know exactly what you are doing, and what the dangers are.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mpm_netware

Description:	Multi-Processing Module implementing an exclusively threaded web server optimized for Novell NetWare
Status:	MPM
Module Identifier:	mpm_netware_module

Summary

This Multi-Processing Module (MPM) implements an exclusively threaded web server that has been optimized for Novell NetWare.

The main thread is responsible for launching child worker threads which listen for connections and serve them when they arrive. Apache always tries to maintain several *spare* or idle worker threads, which stand ready to serve incoming requests. In this way, clients do not need to wait for a new child threads to be spawned before their requests can be served.

The `StartThreads`, `MinSpareThreads`, `MaxSpareThreads`, and `MaxThreads` regulate how the main thread creates worker threads 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 250 simultaneous requests may need to increase `MaxThreads`, while sites with limited memory may need to decrease `MaxThreads` to keep the server from thrashing (spawning and terminating idle threads). More information about tuning process creation is provided in the performance hints documentation.

`MaxRequestsPerChild` controls how frequently the server recycles processes by killing old ones and launching new ones. On the NetWare OS it is highly recommended that this directive remain set to 0. This allows worker threads to continue servicing requests indefinitely.

See also: Setting which addresses and ports Apache uses.

Directives

- Listen
- ListenBacklog
- MaxRequestsPerChild
- MaxSpareThreads
- MaxThreads
- MinSpareThreads
- SendBufferSize
- StartThreads
- ThreadStackSize

MaxSpareThreads Directive

Description:	
Syntax:	MaxSpareThreads *number*
Default:	`MaxSpareThreads 100`
Context:	server config
Status:	MPM
Module:	mpm_netware

The `MaxSpareThreads` directive sets the desired maximum number of *idle* worker threads. An idle worker thread is one which is not handling a request. If there are more than MaxSpareThreads idle, then the main thread will kill off the excess worker threads.

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.

MaxThreads Directive

Description:	
Syntax:	MaxThreads *number*
Default:	`MaxThreads 250`
Context:	server config
Status:	MPM
Module:	mpm_netware

The MaxThreads directive sets the desired maximum number worker threads allowable.

MinSpareThreads Directive

Description:	
Syntax:	MinSpareThreads *number*
Default:	`MinSpareThreads 10`
Context:	server config
Status:	MPM
Module:	mpm_netware

The `MinSpareThreads` directive sets the desired minimum number of *idle* worker threads. An idle worker thread is one which is not handling a request. If there are fewer than MinSpareThreads idle, then the main thread spawns new worker.

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.

StartThreads Directive

Description:	
Syntax:	StartThreads *number*
Default:	`StartThreads 50`
Context:	server config
Status:	MPM
Module:	mpm_netware

The StartThreads directive sets the desired number of worker threads to spawn and startup

ThreadStackSize Directive

Description:	
Syntax:	ThreadStackSize *number*
Default:	`ThreadStackSize 65536`
Context:	server config
Status:	MPM
Module:	mpm_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.

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Apache Module mpm_winnt

Description:	This Multi-Processing Module is optimized for Windows NT.
Status:	MPM
Module Identifier:	mpm_winnt_module

Summary

This Multi-Processing Module (MPM) is the default for the Windows NT operating systems. It uses a single control process which launches a single child process which in turn creates threads to handle requests

Directives

- CoreDumpDirectory
- Listen
- ListenBacklog
- MaxRequestsPerChild
- PidFile
- SendBufferSize
- ThreadsPerChild

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module perchild

Description:	Multi-Processing Module allowing for daemon processes serving requests to be assigned a variety of different userids
Status:	MPM
Module Identifier:	mpm_perchild_module

Summary

This MPM does not currently work on most platforms. Work is ongoing to make it functional.

This Multi-Processing Module (MPM) implements a hybrid multi-process, multi-threaded web server. A fixed number of processes create threads to handle requests. Fluctuations in load are handled by increasing or decreasing the number of threads in each process.

A single control process launches the number of child processes indicated by the `NumServers` directive at server startup. Each child process creates threads as specified in the `StartThreads` directive. The individual threads then listen for connections and serve them when they arrive.

Apache always tries to maintain a pool of *spare* or idle server threads, which stand ready to serve incoming requests. In this way, clients do not need to wait for new threads to be created. For each child process, Apache assesses the number of idle threads and creates or destroys threads to keep this number within the boundaries specified by `MinSpareThreads` and `MaxSpareThreads`. Since this process is very self-regulating, it is rarely necessary to modify these directives from their default values. The maximum number of clients that may be served simultaneously is determined by multiplying the number of server processes that will be created (`NumServers`) by the maximum number of threads created in each process (`MaxThreadsPerChild`).

While the parent process is usually started as root under Unix in order to bind to port 80, the child processes and threads 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.

See also: Setting which addresses and ports Apache uses.

In addition it adds the extra ability to specify that specific processes should serve requests under different userids. These processes can then be associated with specific virtual hosts.

Directives

- AssignUserId
- ChildPerUserId
- CoreDumpDirectory
- Group
- Listen
- ListenBacklog

AssignUserId Directive

Description:	
Syntax:	AssignUserID *user_id group_id*
Context:	virtual host
Status:	MPM
Module:	perchild

Tie a virtual host to a specific child process. Requests addressed to the virtual host where this directive appears will be served by the process running with the specified user and group id.

ChildPerUserId Directive

Description:	
Syntax:	ChildPerUserID *user_id group_id child_id*
Context:	server config
Status:	MPM
Module:	perchild

Specify a user id and group id for a specific child process. The number of children if set by the NumServers directive. For example, the default value for NumServers is 5 and that means children ids 1,2,3,4 and 5 are available for assigment. If a child does not have an associated ChildPerUserID, it inherits the User and Group settings from the main server

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module prefork

Description:	Implements a non-threaded, pre-forking web server
Status:	MPM
Module Identifier:	mpm_prefork_module

Summary

This Multi-Processing Module (MPM) implements a non-threaded, pre-forking web server which handles request in a manner very similar to the default behavior of Apache 1.3 on Unix.

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 `MaxClients` 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.

Directives

- AcceptMutex
- CoreDumpDirectory
- Listen
- ListenBacklog
- LockFile
- MaxRequestsPerChild
- MaxSpareServers
- MaxSpareServers
- MinSpareServers
- MinSpareServers
- PidFile
- ScoreBoardFile
- SendBufferSize

- ServerLimit
- StartServers
- User

See also

- Setting which addresses and ports Apache uses

AcceptMutex Directive

Description:	Method that Apache uses to serialize multiple children accepting requests on network sockets
Syntax:	AcceptMutex default|*method*
Default:	`AcceptMutex default`
Context:	server config
Status:	MPM
Module:	prefork

The `AcceptMutex` directives sets the method that Apache uses to serialize multiple children accepting requests on network sockets. Prior to Apache 2.0, the method was selectable only at compile time. The optimal method to use is highly architecture and platform dependent. For further details, see the performance tuning documentation.

If this directive is set to `default`, then the compile-time selected default will be used. Other possible methods are listed below. Note that not all methods are available on all platforms. If a method is specified which is not available, a message will be written to the error log listing the available methods.

`flock`
: uses the `flock(2)` system call to lock the file defined by the `LockFile` directive.

`fcntl`
: uses the `fnctl(2)` system call to lock the file defined by the `LockFile` directive.

`sysvsem`
: uses SySV-style semaphores to implement the mutex.

`pthread`
: uses POSIX mutexes as implemented by the POSIX Threads (PThreads) specification.

MaxSpareServers Directive

Description:	Maximum number of idle child server processes
Syntax:	MaxSpareServers *number*
Default:	`MaxSpareServers 10`
Context:	server config
Status:	MPM
Module:	prefork

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.

See also

- `MinSpareServers`
- `StartServers`

MinSpareServers Directive

Description:	Minimum number of idle child server processes
Syntax:	MinSpareServers *number*
Default:	`MinSpareServers 5`
Context:	server config
Status:	MPM
Module:	prefork

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`
- `StartServers`

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module worker

Description:	Multi-Processing Module implementing a hybrid multi-threaded multi-process web server
Status:	MPM
Module Identifier:	mpm_worker_module

Summary

This Multi-Processing Module (MPM) implements a hybrid multi-process multi-threaded server. Each process has a fixed number of threads. The server adjusts to handle load by increasing or decreasing the number of processes.

A single control process is responsible for launching child processes. Each child process creates a fixed number of threads as specified in the `ThreadsPerChild` directive. The individual threads then listen for connections and serve them when they arrive.

Apache always tries to maintain a pool of *spare* or idle server threads, which stand ready to serve incoming requests. In this way, clients do not need to wait for a new threads or processes to be created before their requests can be served. Apache assesses the total number of idle threads in all processes, and forks or kills processes to keep this number within the boundaries specified by `MinSpareThreads` and `MaxSpareThreads`. Since this process is very self-regulating, it is rarely necessary to modify these directives from their default values. The maximum number of clients that may be served simultaneously is determined by multiplying the maximum number of server processes that will be created (`MaxClients`) by the number of threads created in each process (`ThreadsPerChild`).

While the parent process is usually started as root under Unix in order to bind to port 80, the child processes and threads 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.

See also: Setting which addresses and ports Apache uses.

Directives

- CoreDumpDirectory
- Group
- Listen
- ListenBacklog
- LockFile
- MaxClients
- MaxRequestsPerChild
- MaxSpareThreads
- MinSpareThreads
- PidFile
- ScoreBoardFile

- SendBufferSize
- ServerLimit
- StartServers
- ThreadLimit
- ThreadsPerChild
- User

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_access

Description:	Provides access control based on client hostname, IP address, or other characteristics of the client request.
Status:	Base
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`
- `Require`

Allow Directive

Description:	Controls which hosts can access an area of the server
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.

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.

Note that the last three examples above match exactly the same set of hosts.

IPv6 addresses and IPv6 subnets can be specified as shown below:

```
Allow from fe80::a00:20ff:fea7:ccea
Allow from fe80::a00:20ff:fea7:ccea/10
```

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.

Deny Directive

Description:	Controls which hosts are denied access to the server
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

Description:	Controls the default access state and the order in which Allow and Deny are evaluated.
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.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_actions

Description:	This module provides for executing CGI scripts based on media type or request method.
Status:	Base
Module Identifier:	actions_module

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

Description:	Activates a CGI script for a particular handler or content-type
Syntax:	Action *action-type cgi-script*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_actions

This directive adds an action, which will activate *cgi-script* when *action-type* is triggered by the request. The *cgi-script* is the URL-path to a resource that has been designated as a CGI script using `ScriptAliase` or `AddHandler`. 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 `/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

Description:	Activates a CGI script for a particular request method.
Syntax:	Script *method cgi-script*
Context:	server config, virtual host, directory
Status:	Base
Module:	mod_actions

This directive adds an action, which will activate *cgi-script* when a file is requested using the method of *method*. The *cgi-script* is the URL-path to a resource that has been designated as a CGI script using `ScriptAliase` or `AddHandler`. The URL and file path of the requested document is sent using the standard CGI PATH_INFO and PATH_TRANSLATED environment variables.

> 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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_alias

Description:	Provides for mapping different parts of the host filesystem in the document tree and for URL redirection
Status:	Base
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` 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.

Directives

- Alias
- AliasMatch
- Redirect
- RedirectMatch
- RedirectPermanent
- RedirectTemp
- ScriptAlias
- ScriptAliasMatch

See also

- `mod_rewrite`
- Mapping URLs to the filesystem

Alias Directive

Description:	Maps URLs to filesystem locations
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.)

AliasMatch Directive

Description:	Maps URLs to filesystem locations using regular expressions
Syntax:	AliasMatch *regex file-path*|*directory-path*
Context:	server config, virtual host
Status:	Base
Module:	mod_alias

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

Description:	Sends an external redirect asking the client to fetch a different URL
Syntax:	Redirect [*status*] *URL-path URL*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_alias

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 Directive

Description:	Sends an external redirect asking the client to fetch a different URL based on a regular expression match of the current URL
Syntax:	RedirectMatch [*status*] *regex URL*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_alias

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
```

RedirectPermanent Directive

Description:	Sends an external permanent redirect asking the client to fetch a different URL
Syntax:	RedirectPermanent *URL-path URL*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_alias

This directive makes the client know that the Redirect is permanent (status 301). Exactly equivalent to `Redirect permanent`.

RedirectTemp Directive

Description:	Sends an external temporary redirect asking the client to fetch a different URL
Syntax:	RedirectTemp *URL-path URL*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_alias

This directive makes the client know that the Redirect is only temporary (status 302). Exactly equivalent to `Redirect temp`.

ScriptAlias Directive

Description:	Maps a URL to a filesystem location and designates the target as a CGI script
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 that will be processed by `mod_cgi`'s cgi-script handler. 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 Directive

Description:	Maps a URL to a filesystem location using a regular expression and designates the target as a CGI script
Syntax:	ScriptAliasMatch *regex file-path*|*directory-path*
Context:	server config, virtual host
Status:	Base
Module:	mod_alias

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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Apache Module mod_asis

Description:	Sends files that contain their own HTTP headers
Status:	Base
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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_auth

Description:	User authentication using text files
Status:	Base
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`. HTTP Digest Authentication is provided by `mod_auth_digest`.

Directives

- AuthAuthoritative
- AuthGroupFile
- AuthUserFile

See also

- `Require`
- `Satisfy`
- `AuthName`
- `AuthType`

AuthAuthoritative Directive

Description:	Sets whether authorization and authentication are passed to lower level modules
Syntax:	AuthAuthoritative on|off
Default:	`AuthAuthoritative on`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Base
Module:	mod_auth

This information has not been updated for Apache 2.0, which uses a different system for module ordering.

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 `auth_dbm`, `mod_auth_msql`, and `mod_auth_anon`. 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`.

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 behaviour.

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`.

AuthGroupFile Directive

Description:	Sets the name of a text file containing the list of user groups for authentication
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.

AuthUserFile Directive

Description:	Sets the name of a text file containing the list of users and passwords for authentication
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 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:

Create a password file 'Filename' with 'username' as the initial ID. It will prompt for the password:

```
htpasswd -c Filename username
```

Adds or modifies in password file 'Filename' the 'username':

```
htpasswd Filename username2
```

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 will be able to download the AuthUserFile.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_auth_anon

Description:	Allows "anonymous" user access to authenticated areas
Status:	Extension
Module Identifier:	auth_anon_module

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 httpd.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

Description:	Specifies userIDs that areallowed access without password verification
Syntax:	Anonymous *user* [*user*] ...
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

Description:	Configures if authorization will fall-through to other methods
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

Description:	Sets whether the password entered will be logged in the error log
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

Description:	Specifies whether blank passwords are allowed
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

Description:	Sets whether the userID field may be empty
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

Description:	Sets whether to check the password field for a correctly formatted email address
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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Apache Module mod_auth_dbm

Description:	Provides for user authentication using DBM files
Status:	Extension
Module Identifier:	auth_dbm_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`.

Directives

- AuthDBMAuthoritative
- AuthDBMGroupFile
- AuthDBMType
- AuthDBMUserFile

See also

- `AuthName`
- `AuthType`
- `Require`
- `Satisfy`

AuthDBMAuthoritative Directive

Description:	Sets whether authentication and authorization will be passwed on to lower level modules
Syntax:	AuthDBMAuthoritative on|off
Default:	`AuthDBMAuthoritative on`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_auth_dbm

This information has not been updated to take into account the new module ordering techniques in Apache 2.0

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`. 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 behaviour.

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.

AuthDBMGroupFile Directive

Description:	Sets the name of the database file containing the list of user groups for authentication
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:

```
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.

AuthDBMType Directive

Description:	Sets the type of database file that is used to store passwords
Syntax:	AuthDBMType default|SDBM|GDBM|DB
Default:	`AuthDBMType default`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_auth_dbm
Compatibility:	Available in version 2.0.30 and later.

Sets the type of database file that is used to store the passwords. The default database type is determined at compile time. The availability of other types of database files also depends on compile-time settings.

It is crucial that whatever program you use to create your password files is configured to use the same type of database.

AuthDBMUserFile Directive

Description:	Sets thename of a database file containing the list of users and passwords for authentication
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.

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Apache Module mod_auth_digest

Description:	User authentication using MD5 Digest Authentication.
Status:	Experimental
Module Identifier:	auth_digest_module

Summary

This module implements HTTP Digest Authentication. However, it has not been extensively tested and is therefore marked experimental.

Directives

- AuthDigestAlgorithm
- AuthDigestDomain
- AuthDigestFile
- AuthDigestGroupFile
- AuthDigestNcCheck
- AuthDigestNonceFormat
- AuthDigestNonceLifetime
- AuthDigestQop

See also

- `AuthName`
- `AuthType`
- `Require`
- `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.

AuthDigestAlgorithm Directive

Description:	Selects the algorithm used to calculate the challenge and response hases in digest authentication
Syntax:	AuthDigestAlgorithm MD5|MD5-sess
Default:	`AuthDigestAlgorithm MD5`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_digest

The `AuthDigestAlgorithm` directive selects the algorithm used to calculate the challenge and response hashes.

***MD5-sess* is not correctly implemented yet**.

AuthDigestDomain Directive

Description:	URIs that are in the same protection space for digest authentication
Syntax:	AuthDigestDomain *URI* [*URI*] ...
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_digest

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. inluding 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.

AuthDigestFile Directive

Description:	Location of the text file containing the list of users and encoded passwords for digest authentication
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

Description:	Name of the text file containing the list of groups for digest authentication
Syntax:	AuthDigestGroupFile *file-path*
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_digest

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.

AuthDigestNcCheck Directive

Description:	Enables or disables checking of the nonce-count sent by the server
Syntax:	AuthDigestNcCheck On|Off
Default:	`AuthDigestNcCheck Off`
Context:	server config
Status:	Experimental
Module:	mod_auth_digest

Not implemented yet.

AuthDigestNonceFormat Directive

Description:	Determines how the nonce is generated
Syntax:	???
Default:	`???`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_digest

Not implemented yet.

AuthDigestNonceLifetime Directive

Description:	How long the server nonce is valid
Syntax:	AuthDigestNonceLifetime *seconds*
Default:	`AuthDigestNonceLifetime 300`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_digest

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.

AuthDigestQop Directive

Description:	Determines the quality-of-protection to use in digest authentication
Syntax:	AuthDigestQop none|auth|auth-int [auth|auth-int]
Default:	`AuthDigestQop auth`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Experimental
Module:	mod_auth_digest

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.**

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Module mod_auth_ldap

This is an authentication module that allows Apache to authenticate HTTP clients using user entries in an LDAP directory.

Status: Extension
Source File: util_ldap.c
Module Identifier: ldap_module
Compatibility: Available in Apache 2.0 and later.

Summary

mod_auth_ldap supports the following features:

- Known to support the OpenLDAP SDK (both 1.x and 2.x), and the iPlanet (Netscape) SDK.
- Complex authorization policies can be implemented by representing the policy with LDAP filters.
- Support for Microsoft FrontPage allows FrontPage users to control access to their webs, while retaining LDAP for user authentication.
- Uses extensive caching of LDAP operations via mod_ldap.
- Support for LDAP over SSL (requires the Netscape SDK) or TLS (requires the OpenLDAP 2.x SDK).

Directives

Contents

Operation

There are two phases in granting access to a user. The first phase is authentication, in which mod_auth_ldap verifies that the user's credentials are valid. This also called the ***search/bind*** phase. The second phase is authorization, in which mod_auth_ldap determines if the authenticated user is allowed access to the resource in question. This is also known as the ***compare*** phase.

The Authentication Phase

During the authentication phase, mod_auth_ldap searches for an entry in the directory that matches the username that the HTTP client passes. If a single unique match is found, then mod_auth_ldap attempts to bind to the directory server using the DN of the entry plus the password provided by the HTTP client. Because it does a search, then a bind, it is often referred to as the search/bind phase. Here are the steps taken during the search/bind phase.

1. Generate a search filter by combining the attribute and filter provided in the `AuthLDAPURL` directive with the username passed by the HTTP client.
2. Search the directory using the generated filter. If the search does not return exactly one entry, deny or decline access.
3. Fetch the distinguished name of the entry retrieved from the search and attempt to bind to the LDAP server using the DN and the password passed by the HTTP client. If the bind is unsuccessful, deny or decline access.

The following directives are used during the search/bind phase

AuthLDAPURL	Specifies the LDAP server, the base DN, the attribute to use in the search, as well as the extra search filter to use.
AuthLDAPBindDN	An optional DN to bind with during the search phase.
AuthLDAPBindPassword	An optional password to bind with during the search phase.

The Authorization Phase

During the authorization phase, mod_auth_ldap attempts to determine if the user is authorized to access the resource. Many of these checks require mod_auth_ldap to do a compare operation on the LDAP server. This is why this phase is often referred to as the compare phase. mod_auth_ldap accepts the following `require` directives to determine if the credentials are acceptable:

- Grant access if there is a `require valid-user` directive.
- Grant access if there is a `require user` directive, and the username in the directive matches the username passed by the client.
- Grant access if there is a `require dn` directive, and the DN in the directive matches the DN fetched from the LDAP directory.
- Grant access if there is a `require group` directive, and the DN fetched from the LDAP directory (or the username passed by the client) occurs in the LDAP group.
- otherwise, deny or decline access

mod_auth_ldap uses the following directives during the compare phase:

`AuthLDAPURL`	The attribute specified in the URL is used in compare operations for the `require user` operation.
`AuthLDAPCompareDNOnServer`	Determines the behavior of the `require dn` directive.
`AuthLDAPGroupAttribute`	Determines the attribute to use for comparisons in the `require group` directive.

`AuthLDAPGroupAttributeIsDN`	Specifies whether to use the user DN or the username when doing comparisons for the `require group` directive.

The require Directives

Apache's `require` directives are used during the authorization phase to ensure that a user is allowed to access a resource.

require valid-user

If this directive exists, mod_auth_ldap grants access to any user that has successfully authenticated during the search/bind phase.

require user

The `require user` directive specifies what usernames can access the resource. Once mod_auth_ldap has retrieved a unique DN from the directory, it does an LDAP compare operation using the username specified in the `require user` to see if that username is part of the just-fetched LDAP entry. Multiple users can be granted access by putting multiple usernames on the line, separated with spaces. If a username has a space in it, then it must be the only user on the line. In this case, multiple users can be granted access by using multiple `require user` directives, with one user per line. For example, with a `AuthLDAPURL` of *ldap://ldap/o=Airius?cn* (i.e., `cn` is used for searches), the following require directives could be used to restrict access:

```
require user Barbara Jenson
require user Fred User
require user Joe Manager
```

Because of the way that mod_auth_ldap handles this directive, Barbara Jenson could sign on as *Barbara Jenson*, *Babs Jenson* or any other `cn` that she has in her LDAP entry. Only the single `require user` line is needed to support all values of the attribute in the user's entry.

If the `uid` attribute was used instead of the `cn` attribute in the URL above, the above three lines could be condensed to

```
require user bjenson fuser jmanager
```

require group

This directive specifies an LDAP group whose members are allowed access. It takes the distinguished name of the LDAP group. For example, assume that the following entry existed in the LDAP directory:

```
dn: cn=Administrators, o=Airius
objectClass: groupOfUniqueNames
uniqueMember: cn=Barbara Jenson, o=Airius
uniqueMember: cn=Fred User, o=Airius
```

The following directive would grant access to both Fred and Barbara:

```
require group cn=Administrators, o=Airius
```

Behavior of this directive is modified by the `AuthLDAPGroupAttribute` and `AuthLDAPGroupAttributeIsDN` directives.

require dn

The `require dn` directive allows the administrator to grant access based on distinguished names. It specifies a DN that must match for access to be granted. If the distinguished name that was retrieved from the directory server matches the distinguished name in the `require dn`, then authorization is granted.

The following directive would grant access to a specific DN:

```
require dn cn=Barbara Jenson, o=Airius
```

Behavior of this directive is modified by the `AuthLDAPCompareDNOnServer` directive.

Examples

- Grant access to anyone who exists in the LDAP directory, using their UID for searches.

  ```
  AuthLDAPURL ldap://ldap1.airius.com:389/ou=People, o=Airius?uid?sub?(objectClass=*)
  require valid-user
  ```
- The next example is the same as above; but with the fields that have useful defaults omitted. Also, note the use of a redundant LDAP server.

  ```
  AuthLDAPURL ldap://ldap1.airius.com ldap2.airius.com/ou=People, o=Airius
  require valid-user
  ```
- The next example is similar to the previous one, but is uses the common name instead of the UID. Note that this could be problematical if multiple people in the directory share the same `cn`, because a search on `cn` ***must*** return exactly one entry. That's why this approach is not recommended: it's a better idea to choose an attribute that is guaranteed unique in your directory, such as `uid`.

  ```
  AuthLDAPURL ldap://ldap.airius.com/ou=People, o=Airius?cn
  require valid-user
  ```
- Grant access to anybody in the Administrators group. The users must authenticate using their UID.

  ```
  AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid
  require group cn=Administrators, o=Airius
  ```
- The next example assumes that everyone at Airius who carries an alphanumeric pager will have an LDAP attribute of `qpagePagerID`. The example will grant access only to people (authenticated via their UID) who have alphanumeric pagers:

  ```
  AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid??(qpagePagerID=*)
  require valid-user
  ```
- The next example demonstrates the power of using filters to accomplish complicated administrative requirements. Without filters, it would have been necessary to create a new LDAP group and ensure that the group's members remain synchronized with the pager users. This becomes trivial with filters. The goal is to grant access to anyone who has a filter, plus grant access to Joe Manager, who doesn't have a pager, but does need to access the same resource:

  ```
  AuthLDAPURL ldap://ldap.airius.com/o=Airius?uid??(|(qpagePagerID=*)(uid=jmanager))
  require valid-user
  ```

  This last may look confusing at first, so it helps to evaluate what the search filter will look like based on who connects, as shown below. The text in blue is the part that is filled in using the attribute specified in the URL. The text in red is the part that is filled in using the filter specified in the URL. The text in green is filled in using the information that is retrieved from the HTTP client. If Fred User connects as *fuser*, the filter would look like

  ```
  (&(|(qpagePagerID=*)(uid=jmanager))(uid=fuser))
  ```

  The above search will only succeed if *fuser* has a pager. When Joe Manager connects as *jmanager*, the filter looks like

  ```
  (&(|(qpagePagerID=*)(uid=jmanager))(uid=jmanager))
  ```

  The above search will succeed whether *jmanager* has a pager or not.

Using TLS

To use TLS, simply set the `AuthLDAPStartTLS` to on. Nothing else needs to be done (other than ensure that your LDAP server is configured for TLS).

Using SSL

If mod_auth_ldap is linked against the Netscape/iPlanet LDAP SDK, it will not talk to any SSL server unless that server has a certificate signed by a known Certificate Authority. As part of the configuration mod_auth_ldap needs to be told where it can find a database containing the known CAs. This database is in the same format as Netscape Communicator's `cert7.db` database. The easiest way to get this file is to start up a fresh copy of Netscape, and grab the resulting `$HOME/.netscape/cert7.db` file.

To specify a secure LDAP server, use *ldaps://* in the `AuthLDAPURL` directive, instead of *ldap://*.

Using Microsoft FrontPage with mod_auth_ldap

Normally, FrontPage uses FrontPage-web-specific user/group files (i.e., the *mod_auth* module) to handle all authentication. Unfortunately, it is not possible to just change to LDAP authentication by adding the proper directives, because it will break the ***Permissions*** forms in the FrontPage client, which attempt to modify the standard text-based authorization files.

Once a FrontPage web has been created, adding LDAP authentication to it is a matter of adding the following directives to ***every*** `.htaccess` file that gets created in the web

```
AuthLDAPURL             the url
AuthLDAPAuthoritative  off
AuthLDAPFrontPageHack  on
```

`AuthLDAPAuthoritative` must be off to allow mod_auth_ldap to decline group authentication so that Apache will fall back to file authentication for checking group membership. This allows the FrontPage-managed group file to be used.

How It Works

FrontPage restricts access to a web by adding the `require valid-user` directive to the `.htaccess` files. If `AuthLDAPFrontPageHack` is not on, the `require valid-user` directive will succeed for any user who is valid ***as far as LDAP is concerned***. This means that anybody who has an entry in the LDAP directory is considered a valid user, whereas FrontPage considers only those people in the local user file to be valid. The purpose of the hack is to force Apache to consult the local user file (which is managed by FrontPage) - instead of LDAP - when handling the `require valid-user` directive.

Once directives have been added as specified above, FrontPage users will be able to perform all management operations from the FrontPage client.

Caveats

- When choosing the LDAP URL, the attribute to use for authentication should be something that will also be valid for putting into a *mod_auth* user file. The user ID is ideal for this.
- When adding users via FrontPage, FrontPage administrators should choose usernames that already exist in the LDAP directory (for obvious reasons). Also, the password that the administrator enters into the form is ignored, since Apache will actually be authenticating against the password in the LDAP database, and not against the password in the local user file. This could cause confusion for web administrators.
- Apache must be compiled with *mod_auth* in order to use FrontPage support. This is because Apache will still use the *mod_auth* group file for determine the extent of a user's access to the FrontPage web.
- The directives must be put in the `.htaccess` files. Attempting to put them inside `<Location>` or `<Directory>` directives won't work. This is because mod_auth_ldap has to be able to grab the `AuthUserFile` directive that is found in FrontPage `.htaccess` files so that it knows where to look for the valid user list. If the mod_auth_ldap directives aren't in the same `.htaccess` file as the FrontPage directives, then the hack won't work, because mod_auth_ldap will never get a chance to process the `.htaccess` file, and won't be able to find the FrontPage-managed user file.

AuthLDAPAuthoritative directive

Syntax: AuthLDAPAuthoritative on|off
Default: `AuthLDAPAuthoritative on`
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

Set to *off* if this module should let other authentication modules attempt to authenticate the user, should authentication with this module fail. Control is only passed on to lower modules if there is no DN or rule that matches the supplied user name (as passed by the client).

AuthLDAPBindDN directive

Syntax: AuthLDAPBindDN *distinguished-name*
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

An optional DN used to bind to the server when searching for entries. If not provided, mod_auth_ldap will use an anonymous bind.

AuthLDAPBindPassword directive

Syntax: AuthLDAPBindPassword *password*
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

A bind password to use in conjunction with the bind DN. Note that the bind password is probably sensitive data, and should be properly protected. You should only use the `AuthLDAPBindDN` and `AuthLDAPBindPassword` if you absolutely need them to search the directory.

AuthLDAPCompareDNOnServer directive

Syntax: AuthLDAPCompareDNOnServer on|off
Default: `AuthLDAPCompareDNOnServer on`
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

When set, mod_auth_ldap will use the LDAP server to compare the DNs. This is the only foolproof way to compare DNs. mod_auth_ldap will search the directory for the DN specified with the `require dn` directive, then, retrieve the DN and compare it with the DN retrieved from the user entry. If this directive is not set, mod_auth_ldap simply does a string comparison. It is possible to get false negatives with this approach, but it is much faster. Note the mod_ldap cache can speed up DN comparison in most situations.

AuthLDAPDereferenceAliases directive

Syntax: AuthLDAPDereferenceAliases never|searching|finding|always
Default: `AuthLDAPDereferenceAliases Always`
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

This directive specifies when mod_auth_ldap will de-reference aliases during LDAP operations. The default is *always*.

AuthLDAPEnabled directive

Syntax: AuthLDAPEnabled on|off
Default: `AuthLDAPEnabled on`
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

Set to *off* to disable mod_auth_ldap in certain directories. This is useful if you have mod_auth_ldap enabled at or near the top of your tree, but want to disable it completely in certain locations.

AuthLDAPFrontPageHack directive

Syntax: AuthLDAPFrontPageHack on|off
Default: `AuthLDAPFronPageHack off`
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

See the section on using Microsoft FrontPage with mod_auth_ldap.

AuthLDAPGroupAttribute directive

Syntax: AuthLDAPGroupAttribute *attribute*
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

This directive specifies which LDAP attributes are used to check for group membership. Multiple attributes can be used by specifying this directive multiple times. If not specified, then mod_auth_ldap uses the `member` and `uniquemember` attributes.

AuthLDAPGroupAttributeIsDN directive

Syntax: AuthLDAPGroupAttributeIsDN on|off
Default: `AuthLDAPGroupAttributeIsDN on`
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

When set, this directive says to use the distinguished name of the client username when checking for group membership. Otherwise, the username will be used. For example, assume that the client sent the username *bjenson*, which corresponds to the LDAP DN *cn=Babs Jenson, o=Airius*. If this directive is set, mod_auth_ldap will check if the group has *cn=Babs Jenson, o=Airius* as a member. If this directive is not set, then mod_auth_ldap will check if the group has *bjenson* as a member.

AuthLDAPRemoteUserIsDN directive

Syntax: AuthLDAPRemoteUserIsDN on|off
Default: `AuthLDAPUserIsDN off`
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

If this directive is set to on, the value of the *REMOTE_USER* environment variable will be set to the full distinguished name of the authenticated user, rather than just the username that was passed by the client. It is turned off by default.

AuthLDAPStartTLS directive

Syntax: AuthLDAPStartTLS on|off
Default: `AuthLDAPStartTLS off`
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

If this directive is set to on, mod_auth_ldap will start a secure TLS session after connecting to the LDAP server. This requires your LDAP server to support TLS.

AuthLDAPUrl directive

Syntax: AuthLDAPUrl *url*
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_auth_ldap

An RFC 2255 URL which specifies the LDAP search parameters to use. The syntax of the URL is

```
ldap://host:port/basedn?attribute?scope?filter
```

ldap	For regular ldap, use the string *ldap*. For secure LDAP, use *ldaps* instead. Secure LDAP is only available if Apache was linked to an LDAP library with SSL support.
host:port	The name/port of the ldap server (defaults to *localhost:389* for *ldap*, and *localhost:636* for *ldaps*). To specify multiple, redundant LDAP servers, just list all servers, separated by spaces. mod_auth_ldap will try connecting to each server in turn, until it makes a successful connection. Once a connection has been made to a server, that connection remains active for the life of the *httpd* process, or until the LDAP server goes down. If the LDAP server goes down and breaks an existing connection, mod_auth_ldap will attempt to re-connect, starting with the primary server, and trying each redundant server in turn. Note that this is different than a true round-robin search.
basedn	The DN of the branch of the directory where all searches should start from. At the very least, this must be the top of your directory tree, but could also specify a subtree in the directory.
attribute	The attribute to search for. Although RFC 2255 allows a comma-separated list of attributes, only the first attribute will be used, no matter how many are provided. If no attributes are provided, the default is to use `uid`. It's a good idea to choose an attribute that will be unique across all entries in the subtree you will be using.
scope	The scope of the search. Can be either *one* or *sub*. Note that a scope of *base* is also supported by RFC 2255, but is not supported by this module. If the scope is not provided, or if *base* scope is specified, the default is to use a scope of *sub*.
filter	A valid LDAP search filter. If not provided, defaults to `(objectClass=*)`, which will search for all objects in the tree. Filters are limited to approximately 8000 characters (the definition of *MAX_STRING_LEN* in the Apache source code). This should be than sufficient for any application.

When doing searches, the attribute, filter and username passed by the HTTP client are combined to create a search filter that looks like `(&(filter)(attribute=username))`.

For example, consider an URL of *ldap://ldap.airius.com/o=Airius?cn?sub?(posixid=*)*. When a client attempts to connect using a username of *Babs Jenson*, the resulting search filter will be `(&(posixid=*)(cn=Babs Jenson))`.

See above for examples of `AuthLDAPURL` URLs.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_autoindex

Description:	Generates directory indexes, automatically, similar to the Unix *ls* command or the Win32 *dir* shell command
Status:	Base
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 the `FancyIndexing` option is given with 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. These column header links are suppressed with `IndexOptions` directive's `SuppressColumnSorting` option.

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

Autoindex Request Query Arguments

Apache 2.0.23 reorganized the Query Arguments for Column Sorting, and introduced an entire group of new query options. To effectively eliminate all client control over the output, the `IndexOptions IgnoreClient` option was introduced.

The column sorting headers themselves are self-referencing hyperlinks that add the sort query options shown below. Any option below may be added to any request for the directory resource.

- `C=N` sorts the directory by file name
- `C=M` sorts the directory by last-modified date, then file name
- `C=S` sorts the directory by size, then file name
- `C=D` sorts the directory by description, then file name

- `O=A` sorts the listing in Ascending Order
- `O=D` sorts the listing in Descending Order

- `F=0` formats the listing as a simple list (not FancyIndexed)
- `F=1` formats the listing as a FancyIndexed list
- `F=2` formats the listing as an HTMLTable FancyIndexed list

- `V=0` disables version sorting
- `V=1` enables version sorting

- `P=`*`pattern`* lists only files matching the given *pattern*

Note that the 'P'attern query argument is tested *after* the usual IndexIgnore directives are processed, and all file names are still subjected to the same criteria as any other autoindex listing. The Query Arguments parser in mod_autoindex will stop abruptly when an unrecognized option is encountered. The Query Arguments must be well formed, according to the table above.

The simple example below, which can be clipped and saved in a header.html file, illustrates these query options. Note that the unknown "X" argument, for the submit button, is listed last to assure the arguments are all parsed before mod_autoindex encounters the X=Go input.

```
<FORM METHOD="GET">
  Show me a <SELECT NAME="F">
    <OPTION VALUE="0"> Plain list
    <OPTION VALUE="1" SELECTED> Fancy list
    <OPTION VALUE="2"> Table list
  </SELECT>
  Sorted by <SELECT NAME="C">
    <OPTION VALUE="N" SELECTED> Name
    <OPTION VALUE="M"> Date Modified
    <OPTION VALUE="S"> Size
    <OPTION VALUE="D"> Description
  </SELECT>
  <SELECT NAME="O">
    <OPTION VALUE="A" SELECTED> Ascending
    <OPTION VALUE="D"> Descending
  </SELECT>
  <SELECT NAME="V">
    <OPTION VALUE="0" SELECTED> in Normal order
    <OPTION VALUE="1"> in Version order
  </SELECT>
  Matching <INPUT TYPE="text" NAME="P" VALUE="*">
  <INPUT TYPE="submit" NAME="X" VALUE="Go">
</FORM>
```

AddAlt Directive

Description:	Alternate text to display for a file, instead of an icon selected by filename
Syntax:	AddAlt *string file* [*file*] ...
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

`AddAlt` provides 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, has image loading disabled, or fails to retrieve the icon.

Examples:

```
AddAlt "PDF" *.pdf
AddAlt "Compressed" *.gz *.zip *.Z
```

AddAltByEncoding Directive

Description:	Alternate text to display for a file instead of an icon selected by MIME-encoding
Syntax:	AddAltByEncoding *string MIME-encoding* [*MIME-encoding*] ...
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

`AddAltByEncoding` provides 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, has image loading disabled, or fails to retrieve the icon.

Example:

```
AddAltByEncoding "gzip" x-gzip
```

AddAltByType Directive

Description:	Alternate text to display for a file, instead of an icon selected by MIME content-type
Syntax:	AddAltByType *string MIME-type* [*MIME-type*] ...
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

`AddAltByType` 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, has image loading disabled, or fails to retrieve the icon.

Example:

```
AddAltByType "TXT" text/plain
```

AddDescription Directive

Description:	
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 typical, default description field is 23 bytes wide. 6 more bytes are added by the `IndexOptions SuppressIcon` option, 7 bytes are added by the `IndexOptions SuppressSize` option, and 19 bytes are added by the `IndexOptions SuppressLastModified` option. Therefore, the widest default the description column is ever assigned is 55 bytes.

See the DescriptionWidth `IndexOptions` keyword for details on overriding the size of this column, or allowing descriptions of unlimited length.

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

Description:	Icon to display for a file selected by name
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/image.xbm) .gif .jpg .xbm
AddIcon /icons/dir.xbm ^^DIRECTORY^^
AddIcon /icons/backup.xbm *~
```

`AddIconByType` should be used in preference to `AddIcon`, when possible.

AddIconByEncoding Directive

Description:	Icon to display next to files selected by MIME content-encoding
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 `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/compress.xbm x-compress
```

AddIconByType Directive

Description:	Icon to display next to files selected by MIME content-type
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/image.xbm) image/*
```

DefaultIcon Directive

Description:	Icon to display for files when no specific icon is configured
Syntax:	DefaultIcon *url-path*
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
```

HeaderName Directive

Description:	Name of the file that will be inserted at the top of the index listing
Syntax:	HeaderName *filename*
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

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.

Both HeaderName and `ReadmeName` now treat *Filename* as a URI path relative to the one used to access the directory being indexed. *Filename* 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.

IndexIgnore Directive

Description:	Adds to the list of files to hide when listing a directory
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 the replacing the list of ignored files. By default, the list contains `` `.``'. Example:

```
IndexIgnore README .htaccess *~
```

IndexOptions Directive

Description:	Various configuration settings for directory indexing
Syntax:	IndexOptions [+|-]*option* [[+|-]*option*] ...
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

The `IndexOptions` directive specifies the behavior of the directory indexing. *Option* can be one of

DescriptionWidth=[*n* | *] (*Apache 1.3.10 or 2.0.23 and later*)

The `DescriptionWidth` keyword allows you to specify the width of the description column in characters.

`-DescriptionWidth` (or unset) allows mod_autoindex to calculate the best width.

`DescriptionWidth=n` fixes the column width to n bytes wide.

`DescriptionWidth=*` grows the column to the width necessary to accommodate the longest description string.

See the section on `AddDescription` for dangers inherent in truncating descriptions.

FancyIndexing

This turns on fancy indexing of directories.

FoldersFirst (*Apache 1.3.10 or 2.0.23 and later*)

If this option is enabled, subdirectory listings 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.**

HTMLTable *(Experimental, Apache 2.0.23 and later)*

This experimental option with FancyIndexing constructs a simple table for the fancy directory listing. Note this will confuse older browsers. It is particularly necessary if file names or description text will alternate between left-to-right and right-to-left reading order, as can happen on WinNT or other utf-8 enabled platforms.

IconsAreLinks

This makes the icons part of the anchor for the filename, for fancy indexing.

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.

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.

IgnoreClient

This option causes mod_autoindex to ignore all query variables from the client, including sort order (implies `SuppressColumnSorting`.)

NameWidth=[*n* | *] (*Apache 1.3.2 and later*)

The NameWidth keyword allows you to specify the width of the filename column in bytes.

`-NameWidth` (or unset) allows mod_autoindex to calculate the best width.

`NameWidth=n` fixes the column width to n bytes wide.

`NameWidth=*` grows the column to the necessary width.

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 (*Apache 1.3 and later*)

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. **Prior to Apache 2.0.23, this also disabled parsing the Query Arguments for the sort string.** That behavior is now controlled by IndexOptions IgnoreClient in Apache 2.0.23.

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 behaviour, 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.

SuppressIcon (*Apache 2.0.23 and later*)

This will suppress the icon in fancy indexing listings. Combining both *SuppressIcon* and *SuppressRules* yields proper HTML 3.2 output, which by the final specification prohibits IMG and HR tags from the PRE block (used to format FancyIndexed listings.)

SuppressLastModified

This will suppress the display of the last modification date, in fancy indexing listings.

SuppressRules (*Apache 2.0.23 and later*)

This will suppress the horizontal rule lines (HR tags) in directory listings. Combining both *SuppressIcon* and *SuppressRules* yeilds proper HTML 3.2 output, which by the final specification prohibits IMG and HR tags from the PRE block (used to format FancyIndexed listings.)

SuppressSize

This will suppress the file size in fancy indexing listings.

TrackModified (*Apache 1.3.15 or 2.0.23 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 appropriate stat() results. Some 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. **Changes to the size or date stamp of an existing file will not update the Last-Modified header on all Unix platforms.** If this is a concern, leave this option disabled.

VersionSort (*Apache 2.0a3 and later*)

The VersionSort keyword causes files containing version numbers to sort in a natural way. Strings are sorted as usual, except that substrings of digits in the name and description are compared according to their numeric value. For example:

```
foo-1.7
foo-1.7.2
foo-1.7.12
foo-1.8.2
foo-1.8.2a
foo-1.12
```

If the number starts with a zero, then it is considered to be a fraction:

```
foo-1.001
foo-1.002
foo-1.030
foo-1.04
```

Incremental IndexOptions

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 any '+' or '-' prefixes.

IndexOrderDefault Directive

Description:	Sets the default ordering of the directory index
Syntax:	IndexOrderDefault Ascending|Descending Name|Date|Size|Description
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

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

Description:	
Syntax:	ReadmeName *filename*
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_autoindex

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.

See also `HeaderName`, where this behavior is described in greater detail.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_cache

Description:	Content cache keyed to URIs
Status:	Experimental
Module Identifier:	cache_module

Summary

This module is experimental. Documentation is still under development...

mod_cache implements an RFC 2616 compliant HTTP content cache that can be used to cache either local or proxied content. mod_cache requires the services of one or more storage management modules. Two storage management modules are included in the base Apache distribution: *mod_disk_cache* implements a disk based storage manager (generally used for proxy caching) and *mod_mem_cache* implements an in-memory based storage manager (primarily useful for caching local content).

Content stored and retrived keyed to the URL. Content with access protections is not cached.

Directives

- CacheDefaultExpire
- CacheDisable
- CacheEnable
- CacheIgnoreCacheControl
- CacheIgnoreNoLastMod
- CacheLastModifiedFactor
- CacheMaxExpire
- CacheOn

Sample Configuration

Sample httpd.conf

```
#
# Sample Cache Configuration
#
LoadModule cache_module modules/mod_cache.so
<IfModule mod_cache.c>
CacheOn On

#LoadModule disk_cache_module modules/mod_disk_cache.so
<IfModule mod_disk_cache.c>
CacheRoot c:/cacheroot
CacheEnable disk /
CacheDirLevels 5
CacheDirLength 3
</IfModule>

LoadModule mem_cache_module modules/mod_mem_cache.so
<IfModule mod_mem_cache.c>
CacheEnable mem /
CacheSize 4096
CacheMaxObjectCount 100
CacheMinObjectSize 1
CacheMaxObjectSize 2048
</IfModule>

</IfModule>
```

CacheDefaultExpire Directive

Description:	
Syntax:	
Context:	server config
Status:	Experimental
Module:	mod_cache

The default time in seconds to cache a document.

```
CacheDefaultExpire 86400
```

CacheDisable Directive

Description:	Disable caching of specified URLs by specified storage manager
Syntax:	CacheDisable *cache_type url-string*
Context:	server config
Status:	Experimental
Module:	mod_cache

The `CacheDisable` directive instructs mod_cache to *not* cache urls at or above *url-string*.

Example

```
CacheDisable disk /local_files
```

CacheEnable Directive

Description:	Enable caching specified URLs in a specified storage manager
Syntax:	CacheEnable *cache_type url-string*
Context:	server config
Status:	Experimental
Module:	mod_cache

The `CacheEnable` directive instructs mod_cache to cache urls at or below *url-string*. The cache store is specified with the *cache_type* argument. *cache_type mem* instructs mod_cache to use the in-memory cache storage manager implemented by *mod_mem_cache*. *cache_type disk* instructs mod_cache to use the cache storage manager implemented by *mod_disk_cache* .

```
CacheEnable disk /
CacheEnable mem /manual
```

CacheIgnoreCacheControl Directive

Description:	Ignore requests from the client for uncached content
Syntax:	
Context:	server config
Status:	Experimental
Module:	mod_cache

Ignore requests from the client for uncached content

```
CacheIgnoreNoLastMod
```

CacheIgnoreNoLastMod Directive

Description:	Ignore responses where there is no Last Modified Header
Syntax:	
Context:	server config
Status:	Experimental
Module:	mod_cache

Ignore responses where there is no Last Modified Header

```
CacheIgnoreNoLastMod
```

CacheLastModifiedFactor Directive

Description:	The factor used to estimate the Expires date from the LastModified date
Syntax:	
Context:	server config
Status:	Experimental
Module:	mod_cache

The factor used to estimate the Expires date from the LastModified date.

```
CacheLastModifiedFactor
```

CacheMaxExpire Directive

Description:	The maximum time in seconds to cache a document
Syntax:	
Context:	server config
Status:	Experimental
Module:	mod_cache

The maximum time in seconds to cache a document.

```
CacheMaxExpire 604800
```

CacheOn Directive

Description:	
Syntax:	CacheOn
Context:	server config
Status:	Experimental
Module:	mod_cache

```
CacheOn
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_cern_meta

Description:	CERN httpd metafile semantics
Status:	Extension
Module Identifier:	cern_meta_module

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

- MetaDir
- MetaFiles
- MetaSuffix

MetaDir Directive

Description:	Name of the directory to find CERN-style meta information files
Syntax:	MetaDir *directory*
Default:	`MetaDir .web`
Context:	directory
Status:	Extension
Module:	mod_cern_meta

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.

MetaFiles Directive

Description:	Activates CERN meta-file processing
Syntax:	MetaFiles on|off
Default:	`MetaFiles off`
Context:	directory
Status:	Extension
Module:	mod_cern_meta

Turns on/off Meta file processing on a per-directory basis.

MetaSuffix Directive

Description:	File name suffix for the file containg CERN-style meta information
Syntax:	MetaSuffix *suffix*
Default:	`MetaSuffix .meta`
Context:	directory
Status:	Extension
Module:	mod_cern_meta

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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_cgi

Description:	Execution of CGI scripts
Status:	Base
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.

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.

When using a multi-threaded MPM under unix, the module `mod_cgid` should be used in place of this module. At the user level, the two modules are essentially identical.

Directives

- ScriptLog
- ScriptLogBuffer
- ScriptLogLength

See also

- `Options`
- `ScriptAlias`
- `AddHandler`

CGI Environment variables

The server will set the CGI environment variables as described in the CGI specification, with the following provisions:

PATH_INFO

> This will not be available if the `AcceptPathInfo` directive is explicitly set to `off`. The default behavior, if AcceptPathInfo is not given, is that mod_cgi will accept path info (trailing /more/path/info following the script filename in the URI), while the core server will return a 404 NOT FOUND error for requests with additional path info. Omitting the AcceptPathInfo directive has the same effect as setting it `on` for mod_cgi requests.

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

Description:	Location of the CGI script error logfile
Syntax:	ScriptLog *file-path*
Context:	server config
Status:	Base
Module:	mod_cgi, mod_cgid

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.

ScriptLogBuffer Directive

Description:	Maximum amount of PUT or POST requests that will be recorded in the scriptlog
Syntax:	ScriptLogBuffer *bytes*
Default:	`ScriptLogBuffer 1024`
Context:	server config
Status:	Base
Module:	`mod_cgi`, `mod_cgid`

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.

ScriptLogLength Directive

Description:	Size limit of the CGI script logfile
Syntax:	ScriptLogLength *bytes*
Default:	`ScriptLogLength 10385760`
Context:	server config
Status:	Base
Module:	`mod_cgi`, `mod_cgid`

`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.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_cgid

Description:	Execution of CGI scripts using an external CGI daemon
Status:	Base
Module Identifier:	cgid_module
Compatibility:	Unix threaded MPMs only

Summary

Except for the optimizations and the additional `ScriptSock` directive noted below, mod_cgid behaves similarly to mod_cgi. **See the `mod_cgi` Summary for additional details about Apache and CGI.**

On certain unix operating systems, forking a process from a multi-threaded server is a very expensive operation because the new process will replicate all the threads of the parent process. In order to avoid incurring this expense on each CGI invocation, mod_cgid creates an external daemon that is responsible for forking child processes to run CGI scripts. The main server communicates with this daemon using a unix domain socket.

This module is used by default whenever a multi-threaded MPM is selected during the compilation process. At the user level, this module is identical in configuration and operation to `mod_cgi`. The only exception is the additional directive `ScriptSock` which gives the name of the socket to use for communication with the cgi daemon.

Directives

- ScriptLog
- ScriptLogBuffer
- ScriptLogLength
- ScriptSock

ScriptSock Directive

Description:	
Syntax:	ScriptSock *file-path*
Default:	`ScriptSock logs/cgisock`
Context:	server config
Status:	Base
Module:	mod_cgid

This directive sets the name of the socket to use for communication with the CGI daemon. The socket will be opened using the permissions of the user who starts Apache (usually root). To maintain the security of communications with CGI scripts, it is important that no other user has permission to write in the directory where the socket is located.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_charset_lite

Description:	Specify character set translation or recoding
Status:	Experimental
Module Identifier:	charset_lite_module

Summary

This is an **experimental** module and should be used with care. Experiment with your `mod_charset_lite` configuration to ensure that it performs the desired function.

`mod_charset_lite` allows the administrator to specify the source character set of objects as well as the character set they should be translated into before sending to the client. `mod_charset_lite` does not translate the data itself but instead tells Apache what translation to perform. `mod_charset_lite` is applicable to EBCDIC and ASCII host environments. In an EBCDIC environment, Apache normally translates text content from the code page of the Apache process locale to ISO-8859-1. `mod_charset_lite` can be used to specify that a different translation is to be performed. In an ASCII environment, Apache normally performs no translation, so `mod_charset_lite` is needed in order for any translation to take place.

This module provides a small subset of configuration mechanisms implemented by Russian Apache and its associated `mod_charset`.

Directives

- CharsetDefault
- CharsetOptions
- CharsetSourceEnc

Common Problems

Invalid character set names

The character set name parameters of `CharsetSourceEnc` and `CharsetDefault` must be acceptable to the translation mechanism used by APR on the system where `mod_charset_lite` is deployed. These character set names are not standardized and are usually not the same as the corresponding values used in http headers. Currently, APR can only use iconv(3), so you can easily test your character set names using the iconv(1) program, as follows:

```
iconv -f charsetsourceenc-value -t charsetdefault-value
```

Mismatch between character set of content and translation rules

If the translation rules don't make sense for the content, translation can fail in various ways, including:

- The translation mechanism may return a bad return code, and the connection will be aborted.
- The translation mechanism may silently place special characters (e.g., question marks) in the output buffer when it cannot translate the input buffer.

CharsetDefault Directive

Description:	Charset to translate into
Syntax:	CharsetDefault *charset*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Experimental
Module:	mod_charset_lite

The `CharsetDefault` directive specifies the charset that content in the associated container should be translated to.

The value of the *charset* argument must be accepted as a valid character set name by the character set support in APR. Generally, this means that it must be supported by iconv.

Example

```
<Directory "/export/home/trawick/apacheinst/htdocs/convert">
CharsetSourceEnc UTF-16BE
CharsetDefault ISO8859-1
</Directory>
```

CharsetOptions Directive

Description:	Configures charset tranlation behavior
Syntax:	CharsetOptions *option* [*option*] ...
Default:	`CharsetOptions DebugLevel=0 NoImplicitAdd`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Experimental
Module:	mod_charset_lite

The `CharsetOptions` directive configures certain behaviors of `mod_charset_lite`. *Option* can be one of

DebugLevel=*n*

The `DebugLevel` keyword allows you to specify the level of debug messages generated by `mod_charset_lite`. By default, no messages are generated. This is equivalent to `DebugLevel=0`. With higher numbers, more debug messages are generated, and server performance will be degraded. The actual meanings of the numeric values are described with the definitions of the DBGLVL_ constants near the beginning of `mod_charset_lite.c`.

ImplicitAdd | NoImplicitAdd

The `ImplicitAdd` keyword specifies that `mod_charset_lite` should implicitly insert its filter when the configuration specifies that the character set of content should be translated. If the filter chain is explicitly configured using the AddOutputFilter directive, `NoImplicitAdd` should be specified so that `mod_charset_lite` doesn't add its filter.

CharsetSourceEnc Directive

Description:	Source charset of files
Syntax:	CharsetSourceEnc *charset*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Experimental
Module:	mod_charset_lite

The `CharsetSourceEnc` directive specifies the source charset of files in the associated container.

The value of the *charset* argument must be accepted as a valid character set name by the character set support in APR. Generally, this means that it must be supported by iconv.

example

```
<Directory "/export/home/trawick/apacheinst/htdocs/convert">
CharsetSourceEnc UTF-16BE
CharsetDefault ISO8859-1
</Directory>
```

The character set names in this example work with the iconv translation support in Solaris 8.

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Apache Module mod_dav

Description:	Distributed Authoring and Versioning (WebDAV) functionality
Status:	Extension
Module Identifier:	dav_module

Summary

This module provides class 1 and class 2 WebDAV ('Web-based Distributed Authoring and Versioning') functionality for Apache. This extension to the HTTP protocol allows creating, moving, copying, and deleting resources and collections on a remote web server.

To enable mod_dav, add the following to a container in your `httpd.conf` file:

```
Dav On
```

Also, specify a valid filename for the DAV lock database by adding the following to the global section in your `httpd.conf` file:

```
DavLockDB /tmp/DavLock     (Any web-server writable filename, without
an extension)
```

Directives

- Dav
- DavDepthInfinity
- DavLockDB
- DavMinTimeout

Dav Directive

Description:	Enable WebDAV HTTP methods
Syntax:	Dav on|off
Default:	`Dav off`
Context:	directory
Status:	Extension
Module:	mod_dav

Use the `Dav` directive to enable the WebDAV HTTP methods for the given container. You may wish to add a `<Limit>` clause inside the `<location>` directive to limit access to DAV-enabled locations.

Example

```
DavLockDB /tmp/DavLock

<Location /foo>
Dav On

AuthType Basic
AuthName DAV
AuthUserFile user.passwd

  <LimitExcept GET HEAD OPTIONS>
  require user admin
  </LimitExcept>
</Location>
```

DavDepthInfinity Directive

Description:	Allow PROPFIND, Depth: Infinity requests
Syntax:	DavDepthInfinity on|off
Default:	`DavDepthInfinity off`
Context:	directory
Status:	Extension
Module:	mod_dav

Use the `DavDepthInfinity` directive to allow the processing of PROPFIND requests containing the header 'Depth: Infinity'. Because this type of request could constitute a denial-of-service attack, by default it is not allowed.

DavLockDB Directive

Description:	Location of the DAV lock database
Syntax:	DavLockDB *file-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_dav

Use the `DavLockDB` directive to specify the full path to the lock database, excluding an extension. The default (file system) implementation of mod_dav uses a SDBM database to track user locks. The utility `modules/dav/util/lockview` can be used from the server to display all locks in a lock database.

Example

```
DavLockDB /tmp/DavLock
```

DavMinTimeout Directive

Description:	Minimum amount of time the server holds a lock on a DAV resource
Syntax:	DavMinTimeout *seconds*
Default:	`DavMinTimeout 0`
Context:	directory
Status:	Extension
Module:	mod_dav

When a client requests a DAV resource lock, it can also specify a time when the lock will be automatically removed by the server. This value is only a request, and the server can ignore it or inform the client of an arbitrary value.

Use the `DavMinTimeout` directive to specify, in seconds, the minimum lock timeout to return to a client. Microsoft Web Folders defaults to a timeout of 120 seconds; the `DavMinTimeout` can override this to a higher value (like 600 seconds) to reduce the chance of the client losing the lock due to network latency.

Example

```
<Location /MSWord>
DavMinTimeout 600
</Location>
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_deflate

Description:	Compress content before it is delivered to the client
Status:	experimental
Module Identifier:	deflate_module

Summary

The experimental `mod_deflate` module provides the `DEFLATE` output filter that allows output from your server to be compressed before being sent to the client over the network.

Directives

- DeflateFilterNote
- DeflateMemLevel
- DeflateWindowSize

See also

- `AddOutputFilter`
- `SetOutputFilter`

Enabling Compression

Compression is implemented by the `DEFLATE` filter. The following directive will enable compression for documents in the container where it is placed:

Most popular browsers can not handle compression of all content so you may want to enable the 'gzip-only-text/html' note (see below)

```
SetEnv gzip-only-text/html 1
SetOutputFilter DEFLATE
```

Here is an example of enabling compression for the Apache documentation:

```
<Directory "/your-server-root/manual">
SetEnv gzip-only-text/html 1
SetOutputFilter DEFLATE
</Directory>
```

DeflateFilterNote Directive

Description:	Places the compression ratio in a note for logging
Syntax:	DeflateFilterNote *notename*
Context:	server config
Status:	experimental
Module:	mod_deflate

The `DeflateFilterNote` directive specifies that a note about compression ratios should be attached to the request. The name of the note is the value specified for the directive.

DeflateMemLevel Directive

Description:	Amount of memory available to zlib for compression
Syntax:	DeflateMemLevel *value*
Context:	server config
Status:	experimental
Module:	mod_deflate

The `DeflateMemLevel` directive specifies the amount of memory available to zlib for compression.

DeflateWindowSize Directive

Description:	Zlib compression window size
Syntax:	DeflateWindowSize *value*
Context:	server config
Status:	experimental
Module:	mod_deflate

The `DeflateWindowSize` directive specifies the zlib compression window size.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_dir

Description:	Provides for "trailing slash" redirects and serving directory index files
Status:	Base
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

Description:	List of resources to look for when the client requests a directory
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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_env

Description:	Modifies the environment which is passed to CGI scripts and SSI pages
Status:	Base
Module Identifier:	env_module

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.

Directives

- PassEnv
- SetEnv
- UnsetEnv

See also

- Environment Variables

PassEnv Directive

Description:	Passes environment variables from the shell
Syntax:	PassEnv *env-variable* [*env-variable*] ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_env

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

Description:	Sets environment variables
Syntax:	SetEnv *env-variable value*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_env

Sets an environment variable, which is then passed on to CGI scripts and SSI pages. Example:

```
SetEnv SPECIAL_PATH /foo/bin
```

UnsetEnv Directive

Description:	Removes variables from the environment
Syntax:	UnsetEnv *env-variable* [*env-variable*] ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_env

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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_example

Description:	Illustrates the Apache module API
Status:	Experimental
Module Identifier:	example_module

Summary

This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

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

Description:	Demonstration directive to illustrate the Apache module API
Syntax:	Example
Context:	server config, virtual host, directory, .htaccess
Status:	Experimental
Module:	mod_example

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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_expires

Description:	Generation of `Expires` HTTP headers according to user-specified criteria
Status:	Extension
Module Identifier:	expires_module

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

Description:	Enables generation of Expires headers
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

Description:	Value of the Expires header configured by MIME type
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:

```
# enable expirations
ExpiresActive On
# expire GIF images after a month in the client's cache
ExpiresByType image/gif A2592000
# HTML documents are good for a week from the time they were changed
ExpiresByType text/html M604800
```

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 earlier in this document.

ExpiresDefault Directive

Description:	Default algorithm for calculating expiration time
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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Apache Module mod_ext_filter

Description:	Pass the response body through an external program before delivery to the client
Status:	Experimental
Module Identifier:	ext_filter_module

Summary

This is an **experimental** module and should be used with care. Test your `mod_ext_filter` configuration carefully to ensure that it performs the desired function. You may wish to review this information for background on the Apache filtering model.

`mod_ext_filter` presents a simple and familiar programming model for filters. With this module, a program which reads from stdin and writes to stdout (i.e., a Unix-style filter command) can be a filter for Apache. This filtering mechanism is much slower than using a filter which is specially written for the Apache API and runs inside of the Apache server process, but it does have the following benefits:

- the programming model is much simpler
- any programming/scripting language can be used, provided that it allows the program to read from standard input and write to standard output
- existing programs can be used unmodified as Apache filters

Even when the performance characteristics are not suitable for production use, `mod_ext_filter` can be used as a prototype environment for filters.

Directives

- ExtFilterDefine
- ExtFilterOptions

Examples

Generating HTML from some other type of response

```
# mod_ext_filter directive to define a filter to HTML-ize text/c files
# using the external program /usr/bin/enscript, with the type of the
# result set to text/html
ExtFilterDefine c-to-html mode=output intype=text/c outtype=text/html \
                cmd="/usr/bin/enscript --color -W html -Ec -o - -"

<Directory "/export/home/trawick/apacheinst/htdocs/c">

# core directive to cause the new filter to be run on output
SetOutputFilter c-to-html

# mod_mime directive to set the type of .c files to text/c
AddType text/c .c

# mod_ext_filter directive to set the debug level just high
# enough to see a log message per request showing the configuration
# in force
ExtFilterOptions DebugLevel=1

</Directory>
```

Implementing a content encoding filter

```
# mod_ext_filter directive to define the external filter
ExtFilterDefine gzip mode=output cmd=/bin/gzip

<Location /gzipped>

# core directive to cause the gzip filter to be run on output
SetOutputFilter gzip

# mod_header directive to add "Content-Encoding: gzip" header field
Header set Content-Encoding gzip

</Location>
```

Note: this gzip example is just for the purposes of illustration. Please refer to `mod_deflate` for a practical implementation.

Slowing down the server

```
# mod_ext_filter directive to define a filter which runs everything
# through cat; cat doesn't modify anything; it just introduces extra
# pathlength and consumes more resources
ExtFilterDefine slowdown mode=output cmd=/bin/cat preservescontentlength

<Location />

# core directive to cause the slowdown filter to be run several times on
# output
SetOutputFilter slowdown slowdown slowdown

</Location>
```

ExtFilterDefine Directive

Description:	
Syntax:	ExtFilterDefine *filtername parameters*
Context:	server config
Status:	Experimental
Module:	mod_ext_filter

The `ExtFilterDefine` directive defines the characteristics of an external filter, including the program to run and its arguments.

filtername specifies the name of the filter being defined. This name can then be used in SetOutputFilter directives. It must be unique among all registered filters. *At the present time, no error is reported by the register-filter API, so a problem with duplicate names isn't reported to the user.*

Subsequent parameters can appear in any order and define the external command to run and certain other characteristics. The only required parameter is *cmd=*. These parameters are:

cmd=*cmdline*

: The `cmd=` keyword allows you to specify the external command to run. If there are arguments after the program name, the command line should be surrounded in quotation marks.

mode=*mode*

: *mode* should be *output* for now (the default). In the future, *mode=input* will be used to specify a filter for request bodies.

intype=*imt*

: This parameter specifies the internet media type (i.e., MIME type) of documents which should be filtered. By default, all documents are filtered. If `intype=` is specified, the filter will be disabled for documents of other types.

outtype=*imt*

: This parameter specifies the internet media type (i.e., MIME type) of filtered documents. It is useful when the filter changes the internet media type as part of the filtering operation. By default, the internet media type is unchanged.

PreservesContentLength

: The `PreservesContentLength` keyword specifies that the filter preserves the content length. This is not the default, as most filters change the content length. In the event that the filter doesn't modify the length, this keyword should be specified.

ExtFilterOptions Directive

Description:	
Syntax:	ExtFilterOptions *option* [*option*] ...
Default:	`ExtFilterOptions DebugLevel=0 NoLogStderr`
Context:	directory
Status:	Experimental
Module:	mod_ext_filter

The `ExtFilterOptions` directive specifies special processing options for `mod_ext_filter`. *Option* can be one of

DebugLevel=*n*

: The `DebugLevel` keyword allows you to specify the level of debug messages generated by `mod_ext_filter`. By default, no debug messages are generated. This is equivalent to `DebugLevel=0`. With higher numbers, more debug messages are generated, and server performance will be degraded. The actual meanings of the numeric values are described with the definitions of the DBGLVL_ constants near the beginning of `mod_ext_filter.c`.

: Note: The core directive LogLevel should be used to cause debug messages to be stored in the Apache error log.

LogStderr | NoLogStderr

: The `LogStderr` keyword specifies that messages written to standard error by the external filter program will be saved in the Apache error log. `NoLogStderr` disables this feature.

Example:

```
ExtFilterOptions LogStderr DebugLevel=0
```

Messages written to the filter's standard error will be stored in the Apache error log. No debug messages will be generated by `mod_ext_filter`.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_file_cache

Description:	Caches a static list of files in memory
Status:	Experimental
Module Identifier:	file_cache_module

Summary

This module should be used with care. You can easily create a broken site using mod_file_cache, so read this document carefully.

Caching frequently requested files that change very infrequently is a technique for reducing server load. mod_file_cache provides two techniques for caching frequently requested *static* files. Through configuration directives, you can direct mod_file_cache to either open then mmap()a file, or to pre-open a file and save the file's open *file handle*. Both techniques reduce server load when processing requests for these files by doing part of the work (specifically, the file I/O) for serving the file when the server is started rather than during each request.

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.

This module is an extension of and borrows heavily from the mod_mmap_static module in Apache 1.3.

Directives

- CacheFile
- MMapFile

Using mod_file_cache

`mod_file_cache` caches a list of statically configured files via `MMapFile` or `CacheFile` directives in the main server configuration.

Not all platforms support both directives. For example, Apache on Windows does not currently support the `MMapStatic` directive, while other platforms, like AIX, support both. You will receive an error message in the server error log if you attempt to use an unsupported directive. If given an unsupported directive, the server will start but the file will not be cached. On platforms that support both directives, you should experiment with both to see which works best for you.

MmapFile Directive

The `MmapFile` directive of `mod_file_cache` maps a list of statically configured files into memory through the system call `mmap()`. This system call is available on most modern Unix derivates, 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 (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 modules 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.

CacheFile Directive

The `CacheFile` directive of `mod_file_cache` opens an active *handle* or *file descriptor* to the file (or files) listed in the configuration directive and places these open file handles in the cache. When the file is requested, the server retrieves the handle from the cache and passes it to the sendfile() (or TransmitFile() on Windows), socket API.

Insert more details about sendfile API...

This file handle caching is done once at server start or restart, only. So whenever one of the cached files changes on the filesystem you *have* to restart the server (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.

Note

Don't bother asking for a for a directive which recursively caches all the files in a directory. Try this instead... See the `Include` directive, and consider this command:

```
find /www/htdocs -type f -print \
| sed -e 's/.*/mmapfile &/' > /www/conf/mmap.conf
```

CacheFile Directive

Description:	
Syntax:	CacheFile *file-path* [*file-path*] ...
Context:	server config
Status:	Experimental
Module:	mod_file_cache

The `CacheFile` directive opens handles to one or more files (given as whitespace separated arguments) and places these handles into the cache at server startup time. Handles to cached files are automatically closed on a server shutdown. When the files have changed on the filesystem, the server should be restarted to to re-cache them.

Be careful with the *file-path* 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`.

Example

```
CacheFile /usr/local/apache/htdocs/index.html
```

MMapFile Directive

Description:	
Syntax:	MMapFile *file-path* [*file-path*] ...
Context:	server config
Status:	Experimental
Module:	mod_file_cache

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 send to the server to re-mmap them.

Be careful with the *file-path* 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`.

Example

```
MMapFile /usr/local/apache/htdocs/index.html
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_headers

Description:	Customization of HTTP request and response headers
Status:	Extension
Module Identifier:	headers_module
Compatibility:	RequestHeader is available only in Apache 2.0

Summary

This module provides directives to control and modify HTTP request and response headers. Headers can be merged, replaced or removed.

Directives

- Header
- RequestHeader

Order of Processing

The directives provided by mod_header can occur almost anywhere within the server configuration. They are valid in the main server config and virtual host sections, inside <Directory>, <Location> and <Files> sections, and within .htaccess files.

The 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:

```
RequestHeader append MirrorID "mirror 12"
RequestHeader unset MirrorID
```

This way round, the MirrorID header is not set. If reversed, the MirrorID header is set to "mirror 12".

Example

1. Copy all request headers that begin with "TS" to the response headers:

   ```
   Header echo ^TS*
   ```

2. Add a header, MyHeader, to the response including a timestamp for when the request was received and how long it took

to begin serving the request. This header can be used by the client to intuit load on the server or in isolating bottlenecks between the client and the server.

```
Header add MyHeader "%D %t"
```

results in this header being added to the response:

```
MyHeader: D=3775428 t=991424704447256
```

3. Say hello to Joe

```
Header add MyHeader "Hello Joe. It took %D microseconds for
Apache to serve this request."
```

results in this header being added to the response:

```
MyHeader: Hello Joe. It took D=3775428 microseconds for Apache to
serve this request.
```

4. Conditionally send MyHeader on the response if and only if header "MyRequestHeader" is present on the request. This is useful for constructing headers in response to some client stimulus. Note that this example requires the services of the mod_setenvif module.

```
SetEnvIf MyRequestHeader value HAVE_MyRequestHeader
Header add MyHeader "%D %t mytext" env=HAVE_MyRequestHeader
```

If the header "MyRequestHeader: value" is present on the HTTP request, the response will contain the following header:

```
MyHeader: D=3775428 t=991424704447256 mytext
```

Header Directive

Description:	Configure HTTP response headers
Syntax:	Header set|append|add|unset|echo *header* [*value*]
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_headers

This directive can replace, merge or remove HTTP response headers. The header is modified just after the content handler and output filters are run, allowing outgoing headers to be modified. 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. The *value* may be a format string.
- **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.
- **echo**

Request headers with this name are echoed back in the response headers. *header* may be a regular expression.

This argument is followed by a *header* name, which can include the final colon, but it is not required. Case is ignored for set, append, add and unset. The *header* name for echo is case sensitive and may be a regular expression.

For `add`, `append` and `set` a *value* is specified as the third argument. If *value* contains spaces, it should be surrounded by doublequotes. *value* may be a character string, a string containing format specifiers or a combination of both. The following format specifiers are supported in *value*:

%t:	The time the request was received in Universal Coordinated Time since the epoch (Jan. 1, 1970) measured in microseconds. The value is preceded by "t=".
%D:	The time from when the request was received to the time the headers are sent on the wire. This is a measure of the duration of the request. The value is preceded by "D=".
%{FOOBAR}e:	The contents of the environment variable FOOBAR.

When the `Header` directive is used with the `add`, `append`, or `set` argument, a fourth argument may be used to specify conditions under which the action will be taken. If the environment variable specified in the `env=...` argument exists (or if the environment variable does not exist and `env=!...` is specified) then the action specified by the `Header` directive will take effect. Otherwise, the directive will have no effect on the request.

The Header directives are processed just before the response is sent to the network. These means that it is possible to set and/or override most headers, except for those headers added by the header filter.

RequestHeader Directive

Description:	Configure HTTP request headers
Syntax:	RequestHeader set|append|add|unset *header* [*value*]
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_headers

This directive can replace, merge or remove HTTP request headers. The header is modified just before the content handler is run, allowing incoming headers to be modified. The action it performs is determined by the first argument. This can be one of the following values:

- **set**
  The request header is set, replacing any previous header with this name
- **append**
  The request 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 request 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 request 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.

The `RequestHeader` directive is processed just before the request is run by its handler in the fixup phase. This should allow headers generated by the browser, or by Apache input filters to be overridden or modified.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_imap

Description:	Server-side imagemap processing
Status:	Base
Module Identifier:	imap_module

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

- ImapBase
- ImapDefault
- ImapMenu

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.zyzzyva.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>
```

ImapBase Directive

Description:	Default base for imagemap files
Syntax:	ImapBase map|referer|*URL*
Default:	`ImapBase http://servername/`
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_imap

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/`.

ImapDefault Directive

Description:	Default action when an imagemap is called with coordinates that are not explicitly mapped
Syntax:	ImapDefault error|nocontent|map|referer|*URL*
Default:	`ImapDefault nocontent`
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_imap

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.

ImapMenu Directive

Description:	Action if no coordinates are given when calling an imagemap
Syntax:	ImapMenu none|formatted|semiformatted|unformatted
Context:	server config, virtual host, directory, .htaccess
Override:	Indexes
Status:	Base
Module:	mod_imap

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.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_include

Description:	Server-parsed html documents (Server Side Includes)
Status:	Base
Module Identifier:	include_module

Summary

This module provides a filter which will process files before they are sent to the client. The processing is controlled by specially formated 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.

Directives

- SSIEndTag
- SSIErrorMsg
- SSIStartTag
- SSITimeFormat
- SSIUndefinedEcho
- XBitHack

See also

- `Options`
- `SetOutputFilter`
- `AcceptPathInfo`

Enabling Server-Side Includes

Server Side Includes are implemented by the `INCLUDES` filter. 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
AddOutputFilter INCLUDES .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
```

For backwards compatibility, the `server-parsed` handler also activates the INCLUDES filter. As well, Apache will activate the INCLUDES filter for any document with mime type `text/x-server-parsed-html` or `text/x-server-parsed-html3` (and the resulting output will have the mime type `text/html`).

For more information, see our Tutorial on Server Side Includes.

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.

For example:

```
<!--#exec cgi="/cgi-bin/example.cgi" -->
```

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`. In particular, if you need to pass additional arguments to a CGI program, using the query string, this cannot be done with `exec cgi`, but can be done with `include virtual`, as shown here:

```
<!--#include virtual="/cgi-bin/example.cgi?argument=value"
-->
```

cmd

The server will execute the given string using `/bin/sh`. The include variables are available to the command, in addition to the usual set of CGI variables.

The use of `#include virtual` is almost always prefered to using either `#exec cgi` or `#exec cmd`. The former (`#include virtual`) used the standard Apache sub-request mechanism to include files or scripts. It is much better tested and maintained.

In addition, on some platforms, like Win32, and on unix when using suexec, you cannot pass arguments to a command in an `exec` directive, or otherwise include spaces in the command. Thus, while the following will work under a non-suexec configuration on unix, it will not produce the desired result under Win32, or when running suexec:

```
<!--#exec cmd="perl /path/to/perlscript arg1 arg2" -->
```

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. Therefore, you cannot include files that are outside of the document root, or above the current document in the directory structure. 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.

If the specified URL is a CGI program, the program will be executed and its output inserted in place of the directive in the parsed file. You may include a query string in a CGI url:

```
<!--#include virtual="/cgi-bin/example.cgi?argument=value"
-->
```

`include virtual` should be used in preference to `exec cgi` to include the output of CGI programs into an HTML document.

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. There are no attributes.

For example:

```
<!--#printenv -->
```

The **printenv** element is available only in 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" -->
```

The **set** element is available only in 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

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`, `echo`, 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, *a 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.

PATH_INFO with Server Side Includes

Files processed for server-side includes no longer accept requests with PATH_INFO (trailing pathname information) by default. You can use the `AcceptPathInfo` directive to configure the server to accept requests with PATH_INFO.

SSIEndTag Directive

Description:	Changes the string that mod_include looks for to end an include command.
Syntax:	SSIEndTag *tag*
Default:	`SSIEndTag "-->"`
Context:	server config, virtual host
Override:	FileInfo
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.30 and later.

This directive changes the string that mod_include looks for to mark the end of a include command.

See also

- `SSIStartTag`

SSIErrorMsg Directive

Description:	Changes the error message displayed when there is an error
Syntax:	SSIErrorMsg *message*
Default:	`SSIErrorMsg "[an error occurred while processing this directive]"`
Context:	server config, virtual host, directory, .htaccess
Override:	
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.30 and later.

The SSIErrorMsg directive changes the error message displayed when mod_include encounters an error. For production servers you may consider changing the default error message to `"<!-- Error -->"` so that the message is not presented to the user.

This directive has the same effect as the `<!--#config errmsg=`*`message`*` -->` element.

SSIStartTag Directive

Description:	
Syntax:	Changes the string that mod_include looks for to start an include element
Default:	`SSIStartTag "<!--"`
Context:	server config, virtual host
Override:	
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.30 and later.

This directive changes the string that mod_include looks for to mark an include element to process.

You may want to use this option if have 2 servers parsing the output of a file each processing different commands (possibly at different times).

See also

- `SSIEndTag`

SSITimeFormat Directive

Description:	Configures the format in which date strings are displayed
Syntax:	SSITimeFormat *formatstring*
Default:	`SSITimeFormat "%A, %d-%b-%Y %H:%M:%S %Z"`
Context:	server config, virtual host, directory, .htaccess
Override:	
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.30 and later.

This directive changes the format in which date strings are displayed when echoing DATE environment variables. The *formatstring* is as in strftime(3) from the C standard library.

This directive has the same effect as the `<!--#config timefmt=`*`formatstring`*` -->` element.

SSIUndefinedEcho Directive

Description:	Changes the string that mod_include displays when a variable isn't set.
Syntax:	SSIUndefinedEcho *tag*
Default:	`SSIUndefinedEcho "<!-- undef -->"`
Context:	server config, virtual host
Override:	FileInfo
Status:	Base
Module:	mod_include
Compatibility:	Available in version 2.0.34 and later.

This directive changes the string that mod_include displays when a variable is not set and "echoed".

XBitHack Directive

Description:	Parse SSI directives in files with the execute bit set
Syntax:	XBitHack on|off|full
Default:	`XBitHack off`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Base
Module:	mod_include
Compatibility:	

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 text/html 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 the full option, unless you assure the group-execute bit is unset for every SSI script which might `#include` a CGI or otherwise produces different output on each hit (or could potentially change on subsequent requests).

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_info

Description:	Provides a comprehensive overview of the server configuration
Status:	Extension
Module Identifier:	info_module

Summary

To configure `mod_info`, add the following to your `httpd.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.

Directives

- AddModuleInfo

AddModuleInfo Directive

Description:	Allows additional information to be added to the module information displayed by the server-info handler
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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_isapi

Description:	ISAPI Extensions within Apache for Windows
Status:	Base
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 extention. **Please *do not* post such problems to Apache's lists or bug reporting pages.**

Directives

- ISAPIAppendLogToErrors
- ISAPIAppendLogToQuery
- ISAPIFileChache
- ISAPILogNotSupported
- ISAPIReadAheadBuffer

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 extention, edit the httpd.conf file and add the following line:

```
AddHandler isapi-isa .dll
```

There is no capability within the Apache server to leave a requested module loaded. However, you may preload and keep a specific module loaded by using the following syntax in your httpd.conf:

```
ISAPICacheFile c:/WebWork/Scripts/ISAPI/mytest.dll
```

Whether or not you have preloaded an ISAPI extension, all 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 some "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, the directive `ISAPILogNotSupported Off` exists to quiet this noise.

Some servers, like Microsoft IIS, load the ISAPI extension into the server and keep it loaded until memory usage is too high, or unless configuration options are specified. Apache currently loads and unloads the ISAPI extension each time it is requested, unless the ISAPICacheFile directive is specified. This is inefficient, but Apache's memory model makes this the most effective method. Many ISAPI modules are subtly incompatible with the Apache server, and unloading these modules helps to ensure the stability of the server.

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 2.0 `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

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 prefered.

HSE_REQ_IS_KEEP_CONN

Will return the negotiated Keep-Alive status.

HSE_REQ_SEND_RESPONSE_HEADER_EX

Will behave as documented, although the fKeepConn flag is ignored.

HSE_REQ_IS_CONNECTED

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.

ReadClient 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 well as the ALL_HTTP and ALL_RAW values.

Apache 2.0 `mod_isapi` supports additional features introduced in later versions of the ISAPI specification, as well as limited emulation of async I/O and the TransmitFile semantics. Apache also supports preloading ISAPI .dlls for performance, neither of which were not available under Apache 1.3 mod_isapi.

ISAPIAppendLogToErrors Directive

Description:	Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the error log
Syntax:	ISAPIAppendLogToErrors on|off
Default:	`ISAPIAppendLogToErrors off`
Context:	server config
Status:	Base
Module:	mod_isapi

Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the server error log.

ISAPIAppendLogToQuery Directive

Description:	Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the query field
Syntax:	ISAPIAppendLogToQuery on|off
Default:	`ISAPIAppendLogToQuery off`
Context:	server config
Status:	Base
Module:	mod_isapi

Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the query field (appended to the CustomLog %q component).

ISAPIFileChache Directive

Description:	ISAPI .dll files to be loaded at startup
Syntax:	ISAPIFileCache *file-path* [*file-path*] ...
Context:	server config
Status:	Base
Module:	mod_isapi

Specifies a space-separated list of file names to be loaded when the Apache server is launched, and remain loaded until the server is shut down. This directive may be repeated for every ISAPI .dll file desired. The full path name of each file should be specified.

ISAPILogNotSupported Directive

Description:	Log unsupported feature requests from ISAPI extensions
Syntax:	ISAPILogNotSupported on|off
Default:	`ISAPILogNotSupported on`
Context:	server config
Status:	Base
Module:	mod_isapi

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.

ISAPIReadAheadBuffer Directive

Description:	Size of the Read Ahead Buffer sent to ISAPI extensions
Syntax:	ISAPIReadAheadBuffer *size*
Default:	`ISAPIReadAheadBuffer 49152`
Context:	server config
Status:	Base
Module:	mod_isapi

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.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Module mod_ldap

This module is not included with the standard distribution. It is available via CVS in the http-ldap module.

This module adds LDAP connection pooling and result caching to other Apache LDAP modules (like mod_auth_ldap).

Status: Extension
Source File: util_ldap.c
Module Identifier: ldap_module
Compatibility: Available in Apache 2.0 and later.

Summary

This module was created to improve the performance of websites relying on backend connections to LDAP servers. In addition to the functions provided by the standard LDAP libraries, this module adds an LDAP connection pool and an LDAP shared memory cache.

To enable this module, LDAP support must be compiled into apr-util. This is achieved by adding the --*with-ldap* flag to the `./configure` script when building Apache.

Directives

- `LDAPCacheEntries`
- `LDAPCacheTTL`
- `LDAPCertDBPath`
- `LDAPOpCacheEntries`
- `LDAPOpCacheTTL`
- `LDAPSharedCacheSize`

LDAP Connection Pool

LDAP connections are pooled from request to request. This allows the LDAP server to remain connected and bound ready for the next request, without the need to unbind/connect/rebind. The performance advantages are similar to the effect of HTTP keepalives.

On a busy server it is possible that many requests will try and access the same LDAP server connection simultaneously. Where an LDAP connection is in use, Apache will create a new connection alongside the original one. This ensures that the connection pool does not become a bottleneck.

There is no need to manually enable connection pooling in the Apache configuration. Any module using this module for access to LDAP services will share the connection pool.

LDAP Cache

For improved performance, mod_ldap uses an aggressive caching strategy to minimize the number of times that the LDAP server must be contacted. Caching can easily double or triple the throughput of Apache when it is serving pages protected with mod_auth_ldap. In addition, the load on the LDAP server will be significantly decreased.

mod_ldap supports two types of LDAP caching during the search/bind phase with a ***search/bind cache*** and during the compare phase with two ***operation caches***. Each LDAP URL that is used by the server has its own set of these three caches.

- ### The Search/Bind Cache

  The process of doing a search and then a bind is the most time-consuming aspect of LDAP operation, especially if the directory is large. The search/bind cache is used to cache all searches that resulted in successful binds. Negative results (i.e., unsuccessful searches, or searches that did not result in a successful bind) are not cached. The rationale behind this decision is that connections with invalid credentials are only a tiny percentage of the total number of connections, so by not caching invalid credentials, the size of the cache is reduced.

  mod_ldap stores the username, the DN retrieved, the password used to bind, and the time of the bind in the cache. Whenever a new connection is initiated with the same username, mod_ldap compares the password of the new connection with the password in the cache. If the passwords match, and if the cached entry is not too old, mod_ldap bypasses the search/bind phase.

  The search and bind cache is controlled with the `LDAPCacheEntries` and `LDAPCacheTTL` directives.

- ### Operation Caches

  During attribute and distinguished name comparison functions, mod_ldap uses two operation caches to cache the compare operations. The first compare cache is used to cache the results of compares done to test for LDAP group membership. The second compare cache is used to cache the results of comparisons done between distinguished names.

  The behavior of both of these caches is controlled with the `LDAPOpCacheEntries` and `LDAPOpCacheTTL` directives.

- ### Monitoring the Cache

  mod_ldap has a content handler that allows administrators to monitor the cache performance. The name of the content handler is *ldap-status*, so the following directives could be used to access the mod_ldap cache information:

  ```
  <Location /server/cache-info >
   SetHandler ldap-status
  </Location>
  ```

  By fetching the URL *http://servername/cache-info*, the administrator can get a status report of every cache that is used by mod_ldap cache. Note that if Apache does not support shared memory, then each *httpd* instance has its own cache, so reloading the URL will result in different information each time, depending on which *httpd* instance processes the request.

LDAPSharedCacheSize directive

Syntax: LDAPSharedCacheSize *bytes*
Context: server config
Override: Not Applicable
Status: Extension
Module: mod_ldap

Specifies the number of bytes to specify for the shared memory cache. The default is 100kb.

LDAPCacheEntries directive

Syntax:LDAPCacheEntries *size*
Context: server config
Override: Not Applicable
Status: Extension
Module: mod_ldap

Specifies the maximum size of the primary LDAP cache. This cache contains successful search/binds. Set it to 0 to turn off search/bind caching.

The default size is 1024 cached searches.

LDAPCacheTTL directive

Syntax: LDAPCacheTTL *seconds*
Context: server config
Override: Not Applicable
Status: Extension
Module: mod_ldap

Specifies the time (in seconds) that an item in the search/bind cache remains valid. The default is 600 seconds (10 minutes).

LDAPOpCacheEntries directive

Syntax: LDAPOpCacheEntries *seconds*
Context: server config
Override: Not Applicable
Status: Extension
Module: mod_ldap

This specifies the number of entries mod_ldap will use to cache LDAP compare operations. The default is 1024 entries. Setting it to 0 disables operation caching.

LDAPOpCacheTTL directive

Syntax: LDAPOpCacheTTL *seconds*
Context: server config
Override: Not Applicable
Status: Extension
Module: mod_ldap

Specifies the time (in seconds) that entries in the operation cache remain valid. The default is 600 seconds.

LDAPCertDBPath directive

Syntax: LDAPCertDBPath *directory-path*
Context: server config
Override: Not Applicable
Status: Extension
Module: mod_ldap

This directive is only valid if Apache has been linked against the Netscape/iPlanet Directory SDK.

It specifies in which directory mod_ldap should look for the certificate authorities database for SSL support. There should be a file named `cert7.db` in that directory.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_log_config

Description:	Logging of the requests made to the server
Status:	Base
Module Identifier:	log_config_module

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.

Directives

- CookieLog
- CustomLog
- LogFormat
- TransferLog

See also

- Apache Log Files

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.
%...{Foobar}C:	The contents of cookie "Foobar" in the request sent to the server.
%...D:	The time taken to serve the request, in microseconds.
%...{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.

%...X:	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.`
	(This directive was %...c in late versions of Apache 1.3, but this conflicted with the historical ssl %...{var}c syntax.)

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 Listen 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.

CookieLog Directive

Description:	Sets filename for the logging of cookies
Syntax:	CookieLog *filename*
Context:	server config, virtual host
Status:	Base
Module:	mod_log_config
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

Description:	Sets filename and format of log file
Syntax:	CustomLog *file*|*pipe format*|*nickname* [env=[!]*environment-variable*]
Context:	server config, virtual host
Status:	Base
Module:	mod_log_config
Compatibility:	Nickname only available in Apache 1.3 or later. Conditional logging available in 1.3.5 or later.

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

Description:	Describes a format for use in a log file
Syntax:	LogFormat *format*|*nickname* [*nickname*]
Context:	server config, virtual host
Status:	Base
Module:	mod_log_config
Compatibility:	Nickname only available in Apache 1.3 or later.

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.

For example:

```
LogFormat "%v %h %l %u %t \"%r\" %>s %b" vhost_common
```

TransferLog Directive

Description:	Specifly location of a log file
Syntax:	TransferLog *file*|*pipe*
Context:	server config, virtual host
Status:	Base
Module:	mod_log_config
Compatibility:	

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 specified `LogFormat` directive (which 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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_mime

Description:	Associates the requested filename's extensions with the file's behavior (handlers and filters) and content (mime-type, language, character set and encoding)
Status:	Base
Module Identifier:	mime_module

Summary

This module is used to associate various bits of "meta information" with files by their filename extensions. This information relates the filename of the document to it's mime-type, language, character set and encoding. This information is sent to the browser, and participates in content negotiation, so the user's preferences are respected when choosing one of several possible files to serve. See `mod_negotiation` for more information about content negotiation.

The directives `AddCharset`, `AddEncoding`, `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, 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.

In addition, `mod_mime` may define the handler and filters that originate and process content. The directives `AddHandler`, `AddOutputFilter`, and `AddInputFilter` control the modules or scripts that serve the document. The `MultiviewsMatch` directive allows `mod_negotiation` to consider these file extensions to included when testing Multiviews matches.

While `mod_mime` associates meta-information with filename extensions, the `core` server provides directives that are used to associate all the files in a given container (*e.g.*, `<location>`, `<directory>`, or `<Files>`) with particular meta-information. These directives include `ForceType`, `SetHandler`, `SetInputFilter`, and `SetOutputFilter`. The core directives override any filename extension mappings defined in `mod_mime`.

Note that changing the meta-information for 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. If you change the meta-information (language, content type, character set or encoding) you may need to 'touch' affected files (updating their last modified date) to ensure that all visitors are receive the corrected content headers.

Directives

- RemoveCharset
- RemoveEncoding
- RemoveHandler
- RemoveInputFilter
- RemoveLanguage
- RemoveOutputFilter
- RemoveType
- TypesConfig

See also

- `MimeMagicFile`
- `AddDefaultCharset`
- `ForceType`
- `DefaultType`
- `SetHandler`
- `SetInputFilter`
- `SetOutputFilter`

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. 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.

Content encoding

A file of a particular MIME type can additionally be encoded a particular way to simplify transmission over the Internet. While this usually will refer to compression, such as `gzip`, it can also refer to encryption, such a `pgp` or to an encoding such as UUencoding, which is designed for transmitting a binary file in an ASCII (text) format.

The MIME RFC puts it this way:

> The Content-Encoding entity-header field is used as a modifier to the media-type. When present, its value indicates what additional content coding has been applied to the resource, and thus what decoding mechanism must be applied in order to obtain the media-type referenced by the Content-Type header field. The Content-Encoding is primarily used to allow a document to be compressed without losing the identity of its underlying media type.

By using more than one file extension (see section above about multiple file extensions), you can indicate that a file is of a particular *type*, and also has a particular *encoding*.

For example, you may have a file which is a Microsoft Word document, which is pkzipped to reduce its size. If the `.doc` extension is associated with the Microsoft Word file type, and the `.zip` extension is associated with the pkzip file encoding, then the file `Resume.doc.zip`would be known to be a pkzip'ed Word document.

Apache send a `Content-encoding` header with the resource, in order to tell the client browser about the encoding method.

```
Content-encoding: pkzip
```

Character sets and languages

In addition to file type and the file encoding, another important piece of information is what language a particular document is in, and in what character set the file should be displayed. For example, the document might be written in the Vietnamese alphabet, or in Cyrillic, and should be displayed as such. This information, also, is transmitted in HTTP headers.

The character set, language encoding and mime type are all used in the process of content negotiation (See `mod_negotiation`) to determine which document to give to the client, when there are alternative documents in more than one character set, language, encoding or mime type. All filename extensions associations created with `AddCharset`, `AddEncoding`, `AddLanguage` and `AddType` directives (and extensions listed in the `MimeMagicFile`) participate in this select process. Filename extensions that are only associated using the `AddHandler`, `AddInputFilter` or `AddOutputFilter` directives may be included or excluded from matching by using the `MultiviewsMatch` directive.

Charset

To convey this further information, Apache optionally sends a `Content-Language` header, to specify the language that the document is in, and can append additional information onto the `Content-Type` header to indicate the particular character set that should be used to correctly render the information.

```
Content-Language: en, fr
Content-Type: text/plain; charset=ISO-8859-2
```

The language specification is the two-letter abbreviation for the language. The `charset` is the name of the particular character set which should be used.

AddCharset Directive

Description:	Maps the given filename extensions to the specified content charset
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
- AddDefaultCharset

AddEncoding Directive

Description:	Maps the given filename extensions to the specified encoding type
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.

AddHandler Directive

Description:	Maps the filename extensions to the specified handler
Syntax:	AddHandler *handler-name extension* [*extension*] ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	

Files having the named *extension* will be served by the specified 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

- SetHandler

AddInputFilter Directive

Description:	Maps filename extensions to the filters that will process client requests
Syntax:	AddInputFilter *filter*[;*filter*...] extension [*extension* ...]
Context:	server config, virtual host, directory, .htaccess
Status:	Base
Module:	mod_mime
Compatibility:	AddInputFilter is only available in Apache 2.0.26 and later.

AddInputFilter maps the filename extensions *extension* to the filters which will process client requests and POST input when they are received by the server. This is in addition to any filters defined elsewhere, including the SetInputFilter directive. This mapping is merged over any already in force, overriding any mappings that already exist for the same *extension*.

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content. Both the filter and *extension* arguments are case-insensitive, and the extension may be specified with or without a leading dot.

AddLanguage Directive

Description:	Maps the given filename extension to the specified content language
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

- `mod_negotiation`

AddOutputFilter Directive

Description:	maps the filename extensions to the filters that will process responses from the server
Syntax:	AddOutputFilter *filter*[;*filter*...] extension [*extension* ...]
Context:	server config, virtual host, directory, .htaccess
Override:	
Status:	Base
Module:	mod_mime
Compatibility:	AddOutputFilter is only available in Apache 2.0.26 and later.

The `AddOutputFilter` directive maps the filename extensions *extension* to the filters which will process responses from the server before they are sent to the client. This is in addition to any filters defined elsewhere, including the `SetOutputFilter` directive. This mapping is merged over any already in force, overriding any mappings that already exist for the same *extension*.

For example, the following configuration will process all .shtml files for server-side includes and will then compress the output using `mod_deflate`.

```
AddOutputFilter INCLUDES;DEFLATE shtml
```

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content. Both the filter and *extension* arguments are case-insensitive, and the extension may be specified with or without a leading dot.

AddType Directive

Description:	Maps the given filename extensions onto the specified content type
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.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

See also

- `DefaultType`
- `ForceType`

DefaultLanguage Directive

Description:	Sets all files in the given scope to the specified language
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.

Example

```
DeafaultLanguage en
```

See also

- `mod_negotiation`

MultiviewsMatch Directive

Description:	The types of files that will be included when searching for a matching file with MultiViews
Syntax:	MultiviewsMatch *[NegotiatedOnly] [Handlers] [Filters] [Any]*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_mime
Compatibility:	Available in Apache 2.0.26 and later.

MultiviewsMatch permits three different behaviors for mod_negotiation's Multiviews feature. Multiviews allows a request for a file, e.g. index.html, to match any negotiated extensions following the base request, e.g. index.html.en, index.html,fr, or index.html.gz.

The NegotiatedOnly option provides that every extension following the base name must correlate to a recognized mod_mime extension for content negotation, e.g. Charset, Content-Type, Language, or Encoding. This is the strictest implementation with the fewest unexpected side effects, and is the default behavior.

To include extensions associated with Handlers and/or Filters, set the MultiviewsMatch directive to either Handlers, Filters, or both option keywords. If all other factors are equal, the smallest file will be served, e.g. in deciding between index.html.cgi of 500 characters and index.html.pl of 1000 bytes, the .cgi file would win in this example. Users of .asis files might prefer to use the Handler option, if .asis files are associated with the asis-handler.

You may finally allow Any extensions to match, even if mod_mime doesn't recognize the extension. This was the behavior in Apache 1.3, and can cause unpredicatable results, such as serving .old or .bak files the webmaster never expected to be served.

For example, the following configuration will allow handlers and filters to participate in Multviews, but will exclude unknown files:

```
MultiviewsMatch Handlers Filters
```

See also

- `Options`

RemoveCharset Directive

Description:	Removes any character set associations for a set of file extensions
Syntax:	RemoveCharset *extension* [*extension*] ...
Context:	directory, .htaccess
Status:	Base
Module:	mod_mime
Compatibility:	RemoveCharset is only available in Apache 2.0.24 and later.

The `RemoveCharset` directive removes any character set 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.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

Example

```
RemoveCharset .html .shtml
```

RemoveEncoding Directive

Description:	Removes any content encoding associations for a set of file extensions
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 be marked 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

Description:	Removes any handler associations for a set of file extensions
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.

RemoveInputFilter Directive

Description:	Removes any input filter associations for a set of file extensions
Syntax:	RemoveInputFilter *extension* [*extension*] ...
Context:	directory, .htaccess
Status:	Base
Module:	mod_mime
Compatibility:	RemoveInputFilter is only available in Apache 2.0.26 and later.

The `RemoveInputFilter` directive removes any input filter 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.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

RemoveLanguage Directive

Description:	Removes any language associations for a set of file extensions
Syntax:	RemoveLanguage *extension* [*extension*] ...
Context:	directory, .htaccess
Status:	Base
Module:	mod_mime
Compatibility:	RemoveLanguage is only available in Apache 2.0.24 and later.

The `RemoveLanguage` directive removes any language 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.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

RemoveOutputFilter Directive

Description:	Removes any output filter associations for a set of file extensions
Syntax:	RemoveOutputFilter *extension* [*extension*] ...
Context:	directory, .htaccess
Override:	
Status:	Base
Module:	mod_mime
Compatibility:	RemoveOutputFilter is only available in Apache 2.0.26 and later.

The `RemoveOutputFilter` directive removes any output filter 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.

The *extension* argument is case-insensitive, and can be specified with or without a leading dot.

RemoveType Directive

Description:	Removes any content type associations for a set of file extensions
Syntax:	RemoveType *extension* [*extension*] ...
Context:	directory, .htaccess
Override:	
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.

TypesConfig Directive

Description:	The location of the mime.types file
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. Most administrators use the provided `mime.types` file, which associates common filename extensions with IANA registered content types. The current list is maintained at `http://www.isi.edu/in-notes/iana/assignments/media-types/media-types`. This simplifies the `httpd.conf` file by providing the majority of media-type definitions, and may be overridden by `AddType` directives as needed. You should not edit the `mime.types` file, because it may be replaced when you upgrade your server.

The file contains lines in the format of the arguments to an `AddType` directive:

```
MIME-type extension extension ...
```

The case of the extension does not matter. Blank lines, and lines beginning with a hash character (`#') are ignored.

Please do not send requests to the Apache HTTP Server Project to add any new entries in the distributed mime.types file unless (1) they are already registered with IANA, and (2) they use widely accepted, non-conflicting filename extensions across platforms. category/x-subtype requests will be automatically rejected, as will any new two-letter extensions as they will likely conflict later with the already crowded language and character set namespace.

See also

- `mod_mime_magic`

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_mime_magic

Description:	Determines the MIME type of a file by looking at a few bytes of its contents
Status:	Extension
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	
	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          \<MIFFile       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 Directive

Description:	Enable MIME-type determination based on file contents using the specified magic file
Syntax:	MimeMagicFile *file-path*
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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_negotiation

Description:	Provides for content negotiation
Status:	Base
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 `Options`, where the server does an implicit filename pattern match, and choose from amongst the results.

Directives

- CacheNegotiatedDocs
- ForceLangaugePriority
- LanguagePriority

See also

- `DefaultLangauge`
- `AddEncoding`
- `AddLanguage`
- `AddType`

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 `Options`. 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

Description:	Allows content-negotiated documents to be cached by proxy servers
Syntax:	CacheNegotiatedDocs on|off
Default:	`CacheNegotiatedDocs off`
Context:	server config
Status:	Base
Module:	mod_negotiation
Compatibility:	The syntax changed in version 2.0.

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.

Prior to version 2.0, `CacheNegotiatedDocs` did not take an argument; it was turned on by the presence of the directive by itself.

ForceLangaugePriority Directive

Description:	Action to take if a single acceptable document is not found
Syntax:	ForceLanguagePriority None|Prefer|Fallback [Prefer|Fallback]
Default:	`ForceLangaugePriority None`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_negotiation
Compatibility:	Available in version 2.0.30 and later

The `ForceLanguagePriority` directive uses the given [`LanguagePriority`] to satisfy negotiation where the server could otherwise not return a single matching document.

`ForceLanguagePriority Prefer` uses `LanguagePriority` to serve a one valid result, rather than returning an HTTP result 300 (MULTIPLE CHOICES) when there are several equally valid choices. If the directives below were given, and the user's Accept-Language header assigned en and de each as quality .500 (equally acceptable) then then first matching variant, en, will be served.

```
LanguagePriority en fr de
ForceLanguagePriority Prefer
```

`ForceLanguagePriority Fallback` uses `LanguagePriority` to serve a valid result, rather than returning an HTTP result 406 (NOT ACCEPTABLE). If the directives below were given, and the user's Accept-Language only permitted an es langauge response, but such a variant isn't found, then the first variant from the LanguagePriority list below will be served.

```
LanguagePriority en fr de
ForceLanguagePriority Fallback
```

Both options, Prefer and Fallback, may be specified, so either the first matching variant from LanguagePriority will be served if more that one variant is acceptable, or first available document will be served if none of the variants matched the client's acceptable list of languages.

LanguagePriority Directive

Description:	The precendence of language variants for cases where the client does not express a preference
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 or the `ForceLanguagePriority` directive is not `None`. Correctly implemented HTTP/1.1 requests will mean this directive has no effect.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_proxy

Description:	HTTP/1.1 proxy/gateway server
Status:	Extension
Module Identifier:	proxy_module

Summary

Warning

This document has been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be inaccurate, please use it with care.

This module implements a proxy/gateway for Apache. It implements proxying capability for `FTP`, `CONNECT` (for SSL), `HTTP/0.9`, `HTTP/1.0`, and `HTTP/1.1`. 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. Improvements and bugfixes were made in Apache v1.2.x and Apache v1.3.x, then the module underwent a major overhaul for Apache v2.0. The protocol support was upgraded to HTTP/1.1, and filter support was enabled.

Please note that the **caching** function present in mod_proxy up to Apache v1.3.x has been **removed** from mod_proxy and will be incorporated into a new module, mod_cache.

Do not enable proxying with `ProxyRequests` until you have secured your server. Open proxy servers are dangerous both to your network and to the Internet at large.

Directives

- AllowCONNECT
- NoProxy
- ProxyBlock
- ProxyDomain
- ProxyErrorOverride
- ProxyMaxForwards
- ProxyPass
- ProxyPassReverse
- ProxyPreserveHost
- ProxyReceiveBufferSize
- ProxyRemote
- ProxyRequests
- ProxyTimeout
- ProxyVia

Common configuration topics

Forward and Reverse Proxies

Apache can be configured in both a *forward* and *reverse* proxy configuration.

A *forward proxy* is an intermediate system that enables a browser to connect to a remote network to which it normally does not have access. A forward proxy can also be used to cache data, reducing load on the networks between the forward proxy and the remote webserver.

Apache's mod_proxy can be figured to behave like a forward proxy using the `ProxyRemote` directive. In addition, caching of data can be achieved by configuring Apache `mod_cache`. Other dedicated forward proxy packages include Squid.

A *reverse proxy* is a webserver system that is capable of serving webpages sourced from other webservers - in addition to webpages on disk or generated dynamically by CGI - making these pages look like they originated at the reverse proxy.

When configured with the mod_cache module the reverse proxy can act as a cache for slower backend webservers. The reverse proxy can also enable advanced URL strategies and management techniques, allowing webpages served using different webserver systems or architectures to coexist inside the same URL space. Reverse proxy systems are also ideal for implementing centralised logging websites with many or diverse website backends. Complex multi-tier webserver systems can be constructed using an Apache mod_proxy frontend and any number of backend webservers.

The reverse proxy is configured using the `ProxyPass` and `ProxyPassReverse` directives. Caching can be enabled using mod_cache as with the forward proxy.

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 all
Allow from 192.168.0
</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.

When configuring a reverse proxy, access control takes on the attributes of the normal server `<directory>` configuration.

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. (FTP Directory listings are always executed in ASCII mode, however.)

How can I access FTP files outside of my home directory?

An FTP URI is interpreted relative to the home directory of the user who is logging in. Alas, to reach higher directory levels you cannot use /../, as the dots are interpreted by the browser and not actually sent to the FTP server. To address this problem, the so called "Squid %2f hack" was implemented in the Apache FTP proxy; it is is a solution which is also used by other popular proxy servers like the Squid Proxy Cache. By prepending /%2f to the path of your request, you can make such a proxy change the FTP starting directory to / (instead of the home directory).

Example: To retrieve the file `/etc/motd`, you would use the URL

```
ftp://user@host/%2f/etc/motd
```

How can I hide the FTP cleartext password in my browser's URL line?

To log in to an FTP server by username and password, Apache uses different strategies. In absense of a user name and password in the URL altogether, Apache sends an anomymous login to the FTP server, i.e.,

```
user: anonymous
password: apache_proxy@
```

This works for all popular FTP servers which are configured for anonymous access.

For a personal login with a specific username, you can embed the user name into the URL, like in: `ftp://username@host/myfile`. If the FTP server asks for a password when given this username (which it should), then Apache will reply with a [401 Authorization required] response, which causes the Browser to pop up the username/password dialog. Upon entering the password, the connection attempt is retried, and if successful, the requested resource is presented. The advantage of this procedure is that your browser does not display the password in cleartext (which it would if you had used `ftp://username:password@host/myfile` in the first place).

Note

The password which is transmitted in such a way is not encrypted on its way. It travels between your browser and the Apache proxy server in a base64-encoded cleartext string, and between the Apache proxy and the FTP server as plaintext. You should therefore think twice before accessing your FTP server via HTTP (or before accessing your personal files via FTP at all!) When using unsecure channels, an eavesdropper might intercept your password on its way.

Why does Apache start more slowly when using the proxy module?

If you're using the `ProxyBlock` directive, 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.

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.

AllowCONNECT Directive

Description:	
Syntax:	AllowCONNECT *port* [*port*] ...
Default:	`AllowCONNECT 443 563`
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

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 overrride this default and allow connections to the listed ports only.

NoProxy Directive

Description:	
Syntax:	NoProxy *Domain*| *SubNet*| *IpAddr*| *Hostname* [*Domain*| *SubNet*| *IpAddr*| *Hostname*] ...
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

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 *Domain*s from *Hostname*s (both syntactically and semantically; a DNS domain can have a DNS A record, too!), *Domain*s are always written with a leading period.
Note: Domain name comparisons are done without regard to the case, and *Domain*s 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.

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 *Domain*s, 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 *Hostname*s 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

ProxyBlock Directive

Description:	
Syntax:	ProxyBlock *|*word*|*host*|*domain* [*word*|*host*|*domain*] ...
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

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.

ProxyDomain Directive

Description:	
Syntax:	ProxyDomain *Domain*
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

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
```

ProxyErrorOverride Directive

Description:	
Syntax:	ProxyErrorOverride On|Off
Default:	`ProxyErrorOverride Off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in version 2.0 and later

This directive is useful for reverse-proxy setups, where you want to have a common look and feel on the error pages seen by the end user. This also allows for included files (via mod_include's SSI) to get the error code and act accordingly (default behavior would display the error page of the proxied server, turning this on shows the SSI Error message).

ProxyMaxForwards Directive

Description:	
Syntax:	ProxyMaxForwards *number*
Default:	`ProxyMaxForwards 10`
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in Apache 2.0 and later

The `ProxyMaxForwards` directive specifies the maximum number of proxies through which a request may pass. This is set to prevent infinite proxy loops, or a DoS attack.

Example

```
ProxyMaxForwards 10
```

ProxyPass Directive

Description:	
Syntax:	ProxyPass [*path*] !|*url*
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

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`>.

The ! directive is useful in situations where you don't want to reverse-proxy a subdirectory. eg.

```
ProxyPass /mirror/foo/i !
ProxyPass /mirror/foo http://foo.com
```

will proxy all requests to /mirror/foo to foo.com EXCEPT requests made to /mirror/foo/i

NB: order is important. you need to put the exclusions BEFORE the general proxypass directive

ProxyPassReverse Directive

Description:	
Syntax:	ProxyPassReverse [*path*] *url*
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

This directive lets Apache adjust the URL in the `Location`, `Content-Location` and `URI` headers on HTTP redirect responses. 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.

ProxyPreserveHost Directive

Description:	
Syntax:	ProxyPreserveHost on|off
Default:	`ProxyPreserveHost Off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in Apache 2.0.31 and later.

When enabled, this option will pass the Host: line from the incoming request to the proxied host, instead of the hostname specified in the proxypass line.

This option should normally be turned 'off'.

ProxyReceiveBufferSize Directive

Description:	
Syntax:	ProxyReceiveBufferSize *bytes*
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

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
```

ProxyRemote Directive

Description:	
Syntax:	ProxyRemote *match remote-server*
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

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.

This option also supports reverse proxy configuration - a backend webserver can be embedded within a virtualhost URL space even if that server is hidden by another forward proxy.

ProxyRequests Directive

Description:	
Syntax:	ProxyRequests on|off
Default:	`ProxyRequests Off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

This allows or prevents Apache from functioning as a forward proxy server. (Setting ProxyRequests to 'off' does not disable use of the `ProxyPass` directive.)

In a typical reverse proxy configuration, this option should be set to 'off'.

> Do not enable proxying with `ProxyRequests` until you have secured your server. Open proxy servers are dangerous both to your network and to the Internet at large.

ProxyTimeout Directive

Description:	
Syntax:	ProxyTimeout *seconds*
Default:	`ProxyTimeout 300`
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy
Compatibility:	Available in Apache 2.0.31 and later

This directive allows a user to specifiy a timeout on proxy requests. This is usefull when you have a slow/buggy appserver which hangs, and you would rather just return a timeout and fail gracefully instead of waiting however long it takes the server to return

ProxyVia Directive

Description:	
Syntax:	ProxyVia on|off|full|block
Default:	`ProxyVia off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_proxy

This directive controls the use of the `Via:` HTTP header by the proxy. Its intended use is to control the flow of 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.

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Apache Module mod_rewrite

Description:	Provides a rule-based rewriting engine to rewrite requested URLs on the fly
Status:	Extension
Module Identifier:	rewrite_module
Compatibility:	Available in Apache 1.3 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
rse@engelschall.com
www.engelschall.com
```

Directives

- RewriteBase
- RewriteCond
- RewriteEngine
- RewriteLock
- RewriteLog
- RewriteLogLevel
- RewriteMap
- RewriteOptions
- RewriteRule

Interal 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.

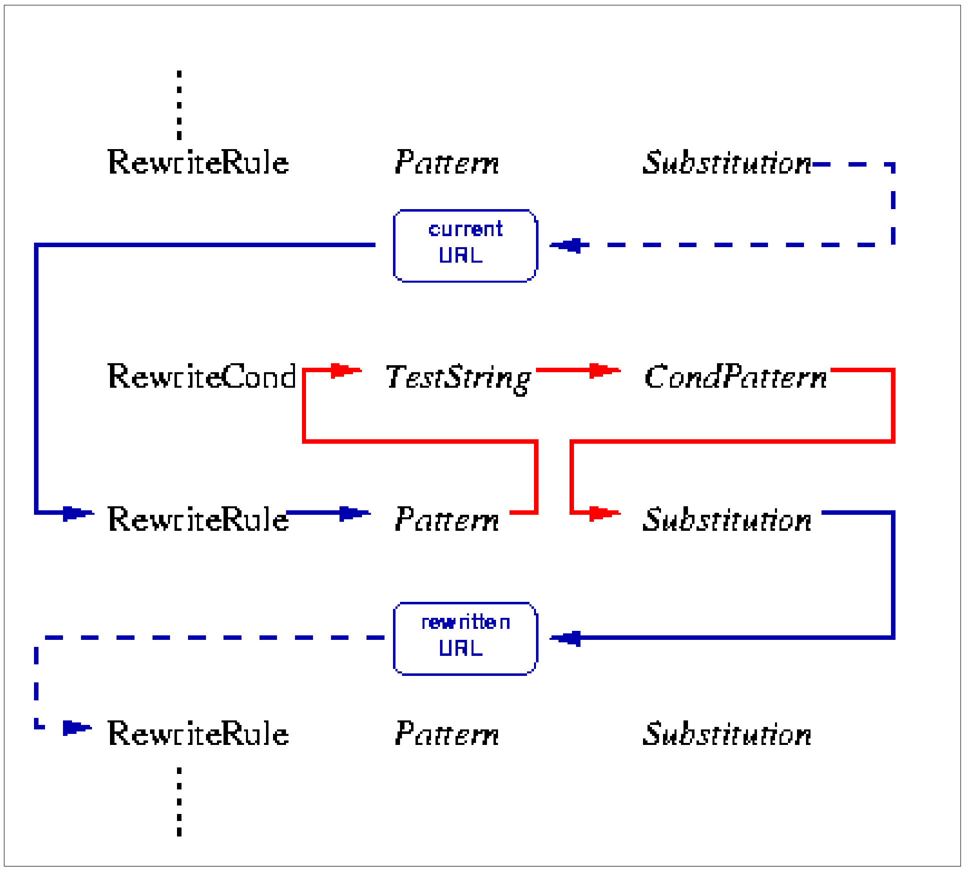


Figure 1: The control flow through the rewriting ruleset

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 slosh ('\') 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 transfered for expansion.

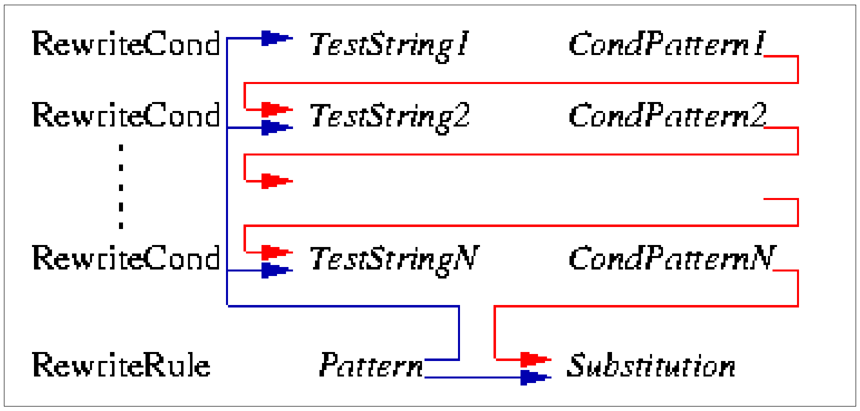


Figure 2: The back-reference flow through a rule.

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.

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.

RewriteBase Directive

Description:	Sets the base URL for per-directory rewrites
Syntax:	RewriteBase *URL-path*
Default:	`See usage for information.`
Context:	directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

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. The default setting is; `RewriteBase` *physical-directory-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.

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.

For 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`.

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 Directive

Description:	Defines a condition under which rewriting will take place
Syntax:	RewriteCond *TestString CondPattern*
Default:	`None`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

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:	**connection & request:**	
HTTP_USER_AGENT HTTP_REFERER HTTP_COOKIE HTTP_FORWARDED HTTP_HOST HTTP_PROXY_CONNECTION HTTP_ACCEPT	REMOTE_ADDR REMOTE_HOST REMOTE_USER REMOTE_IDENT REQUEST_METHOD SCRIPT_FILENAME PATH_INFO QUERY_STRING AUTH_TYPE	
server internals:	**system stuff:**	**specials:**
DOCUMENT_ROOT SERVER_ADMIN SERVER_NAME SERVER_ADDR SERVER_PORT SERVER_PROTOCOL SERVER_SOFTWARE	TIME_YEAR TIME_MON TIME_DAY TIME_HOUR TIME_MIN TIME_SEC TIME_WDAY TIME	API_VERSION THE_REQUEST REQUEST_URI REQUEST_FILENAME IS_SUBREQ

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`**' (**n**o **c**ase)
  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.

RewriteEngine Directive

Description:	Enables or disables runtime rewriting engine
Syntax:	RewriteEngine on|off
Default:	`RewriteEngine off`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

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.

RewriteLock Directive

Description:	Sets the name of the lock file used for RewriteMap synchronization
Syntax:	RewriteLock *file-path*
Default:	`None`
Context:	server config
Status:	Extension
Module:	mod_rewrite

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.

RewriteLog Directive

Description:	Sets the name of the file used for logging rewrite engine processing
Syntax:	RewriteLog *file-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_rewrite

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.

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 Directive

Description:	Sets the verbosity of the log file used by the rewrite engine
Syntax:	RewriteLogLevel *Level*
Default:	`RerwiteLogLevel 0`
Context:	server config, virtual host
Status:	Extension
Module:	mod_rewrite

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.

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
```

RewriteMap Directive

Description:	Defines a mapping function for key-lookup
Syntax:	RewriteMap *MapName* *MapType*:*MapSource*
Default:	`None`
Context:	server config, virtual host
Status:	Extension
Module:	mod_rewrite

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
##

use NDBM_File;
use Fcntl;

($txtmap, $dbmmap) = @ARGV;

open(TXT, "<$txtmap") or die "Couldn't open $txtmap!\n";
tie (%DB, 'NDBM_File', $dbmmap,O_RDWR|O_TRUNC|O_CREAT, 0644) or die "Couldn't create $dbmmap!\n";

while (<TXT>) {
  next if (/^\s*#/ or /^\s*$/);
  $DB{$1} = $2 if (/^\s*(\S+)\s+(\S+)/);
}

untie %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 deadloop! 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!

RewriteOptions Directive

Description:	Sets some special options for the rewrite engine
Syntax:	RewriteOptions *Options*
Default:	`None`
Context:	server config, virtual host, directory, .htaccess
Status:	Extension
Module:	mod_rewrite

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.

RewriteRule Directive

Description:	Defines rules for the rewriting engine
Syntax:	RewriteRule *Pattern Substitution*
Default:	`None`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
Module:	mod_rewrite

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 Nth 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 `$N` to the RewriteRule pattern
2. back-references `%N` to the last matched RewriteCond pattern
3. server-variables as in rule condition test-strings (`%{VARNAME}`)
4. mapping-function calls (`${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 **r**edirect)
  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.

  Note: When you use this flag, make sure that the substitution field is a valid URL! If not, you are redirecting to an invalid location! And remember that this flag itself only prefixes the URL with `http://thishost[:thisport]/`, rewriting continues. Usually you also want to stop and do the redirection immediately. To stop the rewriting you also have to provide the 'L' flag.
- '**`forbidden|F`**' (force URL to be **f**orbidden)
  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 **g**one)
  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 **p**roxy)
  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`**' (**l**ast 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`**' (**n**ext 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`**' (**c**hained 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 **t**ype)
  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 **n**o internal **s**ub-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`**' (**n**o **c**ase)
  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`**' (**q**uery **s**tring **a**ppend)
  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`**' (**n**o URI **e**scaping 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`'.
- '**`passthrough|PT`**' (**p**ass **t**hrough 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`..

  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*' (**s**kip 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 **e**nvironment 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
----------------------------------------------  ----------------------------------
^/somepath(.*) otherpath$1                      not supported, because invalid!

^/somepath(.*) otherpath$1  [R]                 not supported, because invalid!

^/somepath(.*) otherpath$1  [P]                 not supported, because invalid!
----------------------------------------------  ----------------------------------
^/somepath(.*) /otherpath$1                     /otherpath/pathinfo

^/somepath(.*) /otherpath$1 [R]                 http://thishost/otherpath/pathinfo
                                                via external redirection

^/somepath(.*) /otherpath$1 [P]                 not supported, because silly!
----------------------------------------------  ----------------------------------
^/somepath(.*) http://thishost/otherpath$1      /otherpath/pathinfo

^/somepath(.*) http://thishost/otherpath$1 [R]  http://thishost/otherpath/pathinfo
                                                via external redirection

^/somepath(.*) http://thishost/otherpath$1 [P]  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
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_setenvif

Description:	Allows the setting of environment variables based on characteristics of the request
Status:	Base
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
```

Directives

- BrowserMatch
- BrowserMatchNoCase
- SetEnvIf
- SetEnvIfNoCase

See also

- Environment Variables in Apache

BrowserMatch Directive

Description:	Sets environment variables conditional on HTTP User-Agent
Syntax:	BrowserMatch *regex env-variable*[=*value*] [*env-variable*[=*value*]] ...
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 `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

Description:	Sets environment variables conditional on User-Agent without respect to case
Syntax:	BrowserMatchNoCase *regex env-variable*[=*value*] [*env-variable*[=*value*]] ...
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

Description:	Sets environment variables based on attributes of the request
Syntax:	SetEnvIf *attribute regex env-variable*[=*value*] [*env-variable*[=*value*]] ...
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

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.

attribute may be a regular expression when used to match a request header. If *attribute* is a regular expression and it doesn't match any of the request's header names, then *attribute* is not tested against the request's environment variable list.

Example:

```
SetEnvIf Request_URI "\.gif$" object_is_image=gif
SetEnvIf Request_URI "\.jpg$" object_is_image=jpg
SetEnvIf Request_URI "\.xbm$" object_is_image=xbm
:
SetEnvIf Referer www\.mydomain\.com intra_site_referral
:
SetEnvIf object_is_image xbm XBIT_PROCESSING=1
:
SetEnvIf ^TS* ^[a-z].* HAVE_TS
```

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.

The last example will set environment variable `HAVE_TS` if the request contains any headers that begin with "TS" whose values begins with any character in the set [a-z].

SetEnvIfNoCase Directive

Description:	Sets environment variables based on attributes of the request without respect to case
Syntax:	SetEnvIfNoCase *attribute regex env-variable*[=*value*] [*env-variable*[=*value*]] ...
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Base
Module:	mod_setenvif
Compatibility:	Apache 1.3 and above

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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_so

Description:	This module provides for loading of executable code and modules into the server at start-up or restart time.
Status:	Base (Windows>; Optional (Unix)
Module Identifier:	so_module
Compatibility:	Available in Apache 1.3 and later.

Summary

On selected operating systems this module 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), on Windows this may either the `.so` or `.dll` extension. 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.

Warning

Apache 1.3 modules cannot be directly used with Apache 2.0 - the module must be modified to dynamically load or compile into Apache 2.0.

Directives

- LoadFile
- LoadModule

Creating Loadable Modules for Windows

Note

The module name format changed for Windows with Apache 1.3.15 and 2.0 - the modules are now named as mod_foo.so

While mod_so still loads modules with ApacheModuleFoo.dll names, the new naming convention is preferred; if you are converting your loadable module for 2.0, please fix the name to this 2.0 convention.

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 `AP_MODULE_DECLARE_DATA` (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 AP_MODULE_DECLARE_DATA 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 libhttpd.lib export library that is created when the libhttpd.dll shared library is compiled. You may also have to change the compiler settings to ensure that the Apache header files are correctly located. You can find this library in your server root's modules directory. It is best to grab an existing module .dsp file from the tree to assure the build environment is configured correctly, or alternately compare the compiler and link options to your .dsp.

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

Description:	Link in the named object file or library
Syntax:	LoadFile *filename* [*filename*] ...
Default:	`none`
Context:	server config
Status:	Base (Windows>; Optional (Unix)
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.

For example:

```
LoadFile libexex/libxmlparse.so
```

LoadModule Directive

Description:	Links in the object file or library, and adds to the list of active modules
Syntax:	LoadModule *module filename*
Default:	`none`
Context:	server config
Status:	Base (Windows>; Optional (Unix)
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:

```
LoadModule status_module modules/mod_status.so
```

loads the named module from the modules subdirectory of the ServerRoot.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_speling

Description:	Attempts to correct mistaken URLs that users might have entered by ignoring capitalization and by allowing up to one misspelling
Status:	Extension
Module Identifier:	speling_module

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

Description:	Enables the spelling module
Syntax:	CheckSpelling on|off
Default:	`CheckSpelling Off`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Extension
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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_ssl

Description:	Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols
Status:	Extension
Module Identifier:	ssl_module

Summary

This module provides SSL v2/v3 and TLS v1 support for the Apache HTTP Server. It was contributed by Ralf S. Engeschall based on his mod_ssl project and originally derived from work by Ben Laurie.

This module relies on OpenSSL to provide the cryptography engine.

Further details, discussion, and examples are provided in the SSL documentation.

Directives

- SSLCACertificateFile
- SSLCACertificatePath
- SSLCARevocationFile
- SSLCARevocationPath
- SSLCertificateChainFile
- SSLCertificateFile
- SSLCertificateKeyFile
- SSLCipherSuite
- SSLEngine
- SSLLog
- SSLLogLevel
- SSLMutex
- SSLOptions
- SSLPassPhraseDialog
- SSLProtocol
- SSLRandomSeed
- SSLRequire
- SSLRequireSSL
- SSLSessionCache
- SSLSessionCacheTimeout
- SSLVerifyClient
- SSLVerifyDepth

Environment Variables

This module provides a lot of SSL information as additional environment variables to the SSI and CGI namespace. The generated variables are listed in the table below. For backward compatibility the information can be made available under different names, too. Look in the Compatibility chapter for details on the compatibility variables.

Variable Name:	**Value Type:**	**Description:**
`HTTPS`	flag	HTTPS is being used.
`SSL_PROTOCOL`	string	The SSL protocol version (SSLv2, SSLv3, TLSv1)
`SSL_SESSION_ID`	string	The hex-encoded SSL session id
`SSL_CIPHER`	string	The cipher specification name
`SSL_CIPHER_EXPORT`	string	`true` if cipher is an export cipher
`SSL_CIPHER_USEKEYSIZE`	number	Number of cipher bits (actually used)
`SSL_CIPHER_ALGKEYSIZE`	number	Number of cipher bits (possible)
`SSL_VERSION_INTERFACE`	string	The mod_ssl program version
`SSL_VERSION_LIBRARY`	string	The OpenSSL program version
`SSL_CLIENT_M_VERSION`	string	The version of the client certificate
`SSL_CLIENT_M_SERIAL`	string	The serial of the client certificate
`SSL_CLIENT_S_DN`	string	Subject DN in client's certificate
`SSL_CLIENT_S_DN_`*x509*	string	Component of client's Subject DN
`SSL_CLIENT_I_DN`	string	Issuer DN of client's certificate
`SSL_CLIENT_I_DN_`*x509*	string	Component of client's Issuer DN
`SSL_CLIENT_V_START`	string	Validity of client's certificate (start time)
`SSL_CLIENT_V_END`	string	Validity of client's certificate (end time)
`SSL_CLIENT_A_SIG`	string	Algorithm used for the signature of client's certificate
`SSL_CLIENT_A_KEY`	string	Algorithm used for the public key of client's certificate
`SSL_CLIENT_CERT`	string	PEM-encoded client certificate
`SSL_CLIENT_CERT_CHAIN`*n*	string	PEM-encoded certificates in client certificate chain
`SSL_CLIENT_VERIFY`	string	`NONE`, `SUCCESS`, `GENEROUS` or `FAILED:`*reason*
`SSL_SERVER_M_VERSION`	string	The version of the server certificate
`SSL_SERVER_M_SERIAL`	string	The serial of the server certificate
`SSL_SERVER_S_DN`	string	Subject DN in server's certificate
`SSL_SERVER_S_DN_`*x509*	string	Component of server's Subject DN
`SSL_SERVER_I_DN`	string	Issuer DN of server's certificate
`SSL_SERVER_I_DN_`*x509*	string	Component of server's Issuer DN
`SSL_SERVER_V_START`	string	Validity of server's certificate (start time)
`SSL_SERVER_V_END`	string	Validity of server's certificate (end time)
`SSL_SERVER_A_SIG`	string	Algorithm used for the signature of server's certificate
`SSL_SERVER_A_KEY`	string	Algorithm used for the public key of server's certificate
`SSL_SERVER_CERT`	string	PEM-encoded server certificate

[where *x509* is a component of a X.509 DN: `C,ST,L,O,OU,CN,T,I,G,S,D,UID,Email`]

SSI/CGI Environment Variables

Custom Log Formats

When `mod_ssl` is built into Apache or at least loaded (under DSO situation) additional functions exist for the Custom Log Format of `mod_log_config`. First there is an additional ``%{*varname*}x'' eXtension format function which can be used to expand any variables provided by any module, especially those provided by mod_ssl which can you find in the above table.

For backward compatibility there is additionally a special ``%{*name*}c'' cryptography format function provided. Information about this function is provided in the Compatibility chapter.

Example:

```
CustomLog logs/ssl_request_log \ "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\"
%b"
```

SSLCACertificateFile Directive

Description:	File of concatenated PEM-encoded CA Certificates for Client Auth
Syntax:	SSLCACertificateFile *file-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificates of Certification Authorities (CA) whose *clients* you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to `SSLCACertificatePath`.

Example

```
SSLCACertificateFile /usr/local/apache/conf/ssl.crt/ca-bundle-client.crt
```

SSLCACertificatePath Directive

Description:	Directory of PEM-encoded CA Certificates for Client Auth
Syntax:	SSLCACertificatePath *directory-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the Certificates of Certification Authorities (CAs) whose clients you deal with. These are used to verify the client certificate on Client Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named *hash-value*`.N`. And you should always make sure this directory contains the appropriate symbolic links. Use the `Makefile` which comes with mod_ssl to accomplish this task.

Example

```
SSLCACertificatePath /usr/local/apache/conf/ssl.crt/
```

SSLCARevocationFile Directive

Description:	File of concatenated PEM-encoded CA CRLs for Client Auth
Syntax:	SSLCARevocationFile *file-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificate Revocation Lists (CRL) of Certification Authorities (CA) whose *clients* you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded CRL files, in order of preference. This can be used alternatively and/or additionally to `SSLCARevocationPath`.

Example

```
SSLCARevocationFile /usr/local/apache/conf/ssl.crl/ca-bundle-client.crl
```

SSLCARevocationPath Directive

Description:	Directory of PEM-encoded CA CRLs for Client Auth
Syntax:	SSLCARevocationPath *directory-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the directory where you keep the Certificate Revocation Lists (CRL) of Certification Authorities (CAs) whose clients you deal with. These are used to revoke the client certificate on Client Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you have not only to place the CRL files there. Additionally you have to create symbolic links named *hash-value*`.rN`. And you should always make sure this directory contains the appropriate symbolic links. Use the `Makefile` which comes with `mod_ssl` to accomplish this task.

Example
```
SSLCARevocationPath /usr/local/apache/conf/ssl.crl/
```

SSLCertificateChainFile Directive

Description:	File of PEM-encoded Server CA Certificates
Syntax:	SSLCertificateChainFile *file-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the optional *all-in-one* file where you can assemble the certificates of Certification Authorities (CA) which form the certificate chain of the server certificate. This starts with the issuing CA certificate of of the server certificate and can range up to the root CA certificate. Such a file is simply the concatenation of the various PEM-encoded CA Certificate files, usually in certificate chain order.

This should be used alternatively and/or additionally to `SSLCACertificatePath` for explicitly constructing the server certificate chain which is sent to the browser in addition to the server certificate. It is especially useful to avoid conflicts with CA certificates when using client authentication. Because although placing a CA certificate of the server certificate chain into `SSLCACertificatePath` has the same effect for the certificate chain construction, it has the side-effect that client certificates issued by this same CA certificate are also accepted on client authentication. That's usually not one expect.

But be careful: Providing the certificate chain works only if you are using a *single* (either RSA *or* DSA) based server certificate. If you are using a coupled RSA+DSA certificate pair, this will work only if actually both certificates use the *same* certificate chain. Else the browsers will be confused in this situation.

Example
```
SSLCertificateChainFile /usr/local/apache/conf/ssl.crt/ca.crt
```

SSLCertificateFile Directive

Description:	Server PEM-encoded X.509 Certificate file
Syntax:	SSLCertificateFile *file-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive points to the PEM-encoded Certificate file for the server and optionally also to the corresponding RSA or DSA Private Key file for it (contained in the same file). If the contained Private Key is encrypted the Pass Phrase dialog is forced at startup time. This directive can be used up to two times (referencing different filenames) when both a RSA and a DSA based server certificate is used in parallel.

Example
```
SSLCertificateFile /usr/local/apache/conf/ssl.crt/server.crt
```

SSLCertificateKeyFile Directive

Description:	Server PEM-encoded Private Key file
Syntax:	SSLCertificateKeyFile *file-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive points to the PEM-encoded Private Key file for the server. If the Private Key is not combined with the Certificate in the `SSLCertificateFile`, use this additional directive to point to the file with the stand-alone Private Key. When `SSLCertificateFile` is used and the file contains both the Certificate and the Private Key this directive need not be used. But we strongly discourage this practice. Instead we recommend you to separate the Certificate and the Private Key. If the contained Private Key is encrypted, the Pass Phrase dialog is forced at startup time. This directive can be used up to two times (referencing different filenames) when both a RSA and a DSA based private key is used in parallel.

Example

```
SSLCertificateKeyFile /usr/local/apache/conf/ssl.key/server.key
```

SSLCipherSuite Directive

Description:	Cipher Suite available for negotiation in SSL handshake
Syntax:	SSLCipherSuite *cipher-spec*
Default:	`SSLCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP`
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This complex directive uses a colon-separated *cipher-spec* string consisting of OpenSSL cipher specifications to configure the Cipher Suite the client is permitted to negotiate in the SSL handshake phase. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotation with the reconfigured Cipher Suite after the HTTP request was read but before the HTTP response is sent.

An SSL cipher specification in *cipher-spec* is composed of 4 major attributes plus a few extra minor ones:

- *Key Exchange Algorithm*:
  RSA or Diffie-Hellman variants.
- *Authentication Algorithm*:
  RSA, Diffie-Hellman, DSS or none.
- *Cipher/Encryption Algorithm*:
  DES, Triple-DES, RC4, RC2, IDEA or none.
- *MAC Digest Algorithm*:
  MD5, SHA or SHA1.

An SSL cipher can also be an export cipher and is either a SSLv2 or SSLv3/TLSv1 cipher (here TLSv1 is equivalent to SSLv3). To specify which ciphers to use, one can either specify all the Ciphers, one at a time, or use aliases to specify the preference and order for the ciphers (see Table 1).

Tag	**Description**
Key Exchange Algorithm:	
`kRSA`	RSA key exchange
`kDHr`	Diffie-Hellman key exchange with RSA key
`kDHd`	Diffie-Hellman key exchange with DSA key
`kEDH`	Ephemeral (temp.key) Diffie-Hellman key exchange (no cert)
Authentication Algorithm:	
`aNULL`	No authentication
`aRSA`	RSA authentication
`aDSS`	DSS authentication
`aDH`	Diffie-Hellman authentication
Cipher Encoding Algorithm:	
`eNULL`	No encoding
`DES`	DES encoding
`3DES`	Triple-DES encoding
`RC4`	RC4 encoding
`RC2`	RC2 encoding
`IDEA`	IDEA encoding
MAC Digest Algorithm:	
`MD5`	MD5 hash function
`SHA1`	SHA1 hash function
`SHA`	SHA hash function
Aliases:	
`SSLv2`	all SSL version 2.0 ciphers
`SSLv3`	all SSL version 3.0 ciphers
`TLSv1`	all TLS version 1.0 ciphers
`EXP`	all export ciphers
`EXPORT40`	all 40-bit export ciphers only
`EXPORT56`	all 56-bit export ciphers only
`LOW`	all low strength ciphers (no export, single DES)
`MEDIUM`	all ciphers with 128 bit encryption
`HIGH`	all ciphers using Triple-DES
`RSA`	all ciphers using RSA key exchange
`DH`	all ciphers using Diffie-Hellman key exchange
`EDH`	all ciphers using Ephemeral Diffie-Hellman key exchange
`ADH`	all ciphers using Anonymous Diffie-Hellman key exchange
`DSS`	all ciphers using DSS authentication
`NULL`	all ciphers using no encryption

Table 1: OpenSSL Cipher Specification Tags

Now where this becomes interesting is that these can be put together to specify the order and ciphers you wish to use. To speed this up there are also aliases (`SSLv2, SSLv3, TLSv1, EXP, LOW, MEDIUM, HIGH`) for certain groups of ciphers. These tags can be joined together with prefixes to form the *cipher-spec*. Available prefixes are:

- none: add cipher to list
- `+`: add ciphers to list and pull them to current location in list
- `-`: remove cipher from list (can be added later again)
- `!`: kill cipher from list completely (can **not** be added later again)

A simpler way to look at all of this is to use the ``` ``openssl ciphers -v ```" command which provides a nice way to successively create the correct *cipher-spec* string. The default *cipher-spec* string is ``` ``ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP ```" which means the following: first, remove from consideration any ciphers that do not authenticate, i.e. for SSL only the Anonymous Diffie-Hellman ciphers. Next, use ciphers using RC4 and RSA. Next include the high, medium and then the low security ciphers. Finally *pull* all SSLv2 and export ciphers to the end of the list.

```
$ openssl ciphers -v 'ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+SSLv2:+EXP'
NULL-SHA                 SSLv3 Kx=RSA      Au=RSA  Enc=None      Mac=SHA1
NULL-MD5                 SSLv3 Kx=RSA      Au=RSA  Enc=None      Mac=MD5
EDH-RSA-DES-CBC3-SHA     SSLv3 Kx=DH       Au=RSA  Enc=3DES(168) Mac=SHA1
...                      ...               ...     ...           ...
EXP-RC4-MD5              SSLv3 Kx=RSA(512) Au=RSA  Enc=RC4(40)   Mac=MD5  export
EXP-RC2-CBC-MD5          SSLv2 Kx=RSA(512) Au=RSA  Enc=RC2(40)   Mac=MD5  export
EXP-RC4-MD5              SSLv2 Kx=RSA(512) Au=RSA  Enc=RC4(40)   Mac=MD5  export
```

The complete list of particular RSA & DH ciphers for SSL is given in Table 2.

Example

```
SSLCipherSuite RSA:!EXP:!NULL:+HIGH:+MEDIUM:-LOW
```

Cipher-Tag	**Protocol**	**Key Ex.**	**Auth.**	**Enc.**	**MAC**	**Type**
RSA Ciphers:						
`DES-CBC3-SHA`	SSLv3	RSA	RSA	3DES(168)	SHA1	
`DES-CBC3-MD5`	SSLv2	RSA	RSA	3DES(168)	MD5	
`IDEA-CBC-SHA`	SSLv3	RSA	RSA	IDEA(128)	SHA1	
`RC4-SHA`	SSLv3	RSA	RSA	RC4(128)	SHA1	
`RC4-MD5`	SSLv3	RSA	RSA	RC4(128)	MD5	
`IDEA-CBC-MD5`	SSLv2	RSA	RSA	IDEA(128)	MD5	
`RC2-CBC-MD5`	SSLv2	RSA	RSA	RC2(128)	MD5	
`RC4-MD5`	SSLv2	RSA	RSA	RC4(128)	MD5	
`DES-CBC-SHA`	SSLv3	RSA	RSA	DES(56)	SHA1	
`RC4-64-MD5`	SSLv2	RSA	RSA	RC4(64)	MD5	
`DES-CBC-MD5`	SSLv2	RSA	RSA	DES(56)	MD5	
`EXP-DES-CBC-SHA`	SSLv3	RSA(512)	RSA	DES(40)	SHA1	export
`EXP-RC2-CBC-MD5`	SSLv3	RSA(512)	RSA	RC2(40)	MD5	export
`EXP-RC4-MD5`	SSLv3	RSA(512)	RSA	RC4(40)	MD5	export
`EXP-RC2-CBC-MD5`	SSLv2	RSA(512)	RSA	RC2(40)	MD5	export
`EXP-RC4-MD5`	SSLv2	RSA(512)	RSA	RC4(40)	MD5	export
`NULL-SHA`	SSLv3	RSA	RSA	None	SHA1	
`NULL-MD5`	SSLv3	RSA	RSA	None	MD5	
Diffie-Hellman Ciphers:						
`ADH-DES-CBC3-SHA`	SSLv3	DH	None	3DES(168)	SHA1	
`ADH-DES-CBC-SHA`	SSLv3	DH	None	DES(56)	SHA1	
`ADH-RC4-MD5`	SSLv3	DH	None	RC4(128)	MD5	
`EDH-RSA-DES-CBC3-SHA`	SSLv3	DH	RSA	3DES(168)	SHA1	
`EDH-DSS-DES-CBC3-SHA`	SSLv3	DH	DSS	3DES(168)	SHA1	
`EDH-RSA-DES-CBC-SHA`	SSLv3	DH	RSA	DES(56)	SHA1	
`EDH-DSS-DES-CBC-SHA`	SSLv3	DH	DSS	DES(56)	SHA1	
`EXP-EDH-RSA-DES-CBC-SHA`	SSLv3	DH(512)	RSA	DES(40)	SHA1	export
`EXP-EDH-DSS-DES-CBC-SHA`	SSLv3	DH(512)	DSS	DES(40)	SHA1	export
`EXP-ADH-DES-CBC-SHA`	SSLv3	DH(512)	None	DES(40)	SHA1	export
`EXP-ADH-RC4-MD5`	SSLv3	DH(512)	None	RC4(40)	MD5	export

Table 2: Particular SSL Ciphers

SSLEngine Directive

Description:	SSL Engine Operation Switch
Syntax:	SSLEngine on|off
Default:	`SSLEngine off`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive toggles the usage of the SSL/TLS Protocol Engine. This is usually used inside a `<VirtualHost>` section to enable SSL/TLS for a particular virtual host. By default the SSL/TLS Protocol Engine is disabled for both the main server and all configured virtual hosts.

Example

```
<VirtualHost _default_:443>
SSLEngine on
...
</VirtualHost>
```

SSLLog Directive

Description:	Where to write the dedicated SSL engine logfile
Syntax:	SSLLog *file-path*
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the name of the dedicated SSL protocol engine logfile. Error type messages are additionally duplicated to the general Apache error log file (directive `ErrorLog`). Put this somewhere where it cannot be used for symlink attacks on a real server (i.e. somewhere where only root can write). If the *file-path* does not begin with a slash ('`/`') then it is assumed to be relative to the *Server Root*. If *file-path* begins with a bar ('`|`') then the following string is assumed to be a path to an executable program to which a reliable pipe can be established. The directive should occur only once per virtual server config.

Example

```
SSLLog /usr/local/apache/logs/ssl_engine_log
```

SSLLogLevel Directive

Description:	Logging level for the dedicated SSL engine logfile
Syntax:	SSLLogLevel *level*
Default:	`SSLLogLevel none`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the verbosity degree of the dedicated SSL protocol engine logfile. The *level* is one of the following (in ascending order where higher levels include lower levels):

- `none`
  no dedicated SSL logging is done, but messages of level ``error" are still written to the general Apache error logfile.
- `error`
  log messages of error type only, i.e. messages which show fatal situations (processing is stopped). Those messages are also duplicated to the general Apache error logfile.
- `warn`
  log also warning messages, i.e. messages which show non-fatal problems (processing is continued).
- `info`
  log also informational messages, i.e. messages which show major processing steps.
- `trace`
  log also trace messages, i.e. messages which show minor processing steps.
- `debug`
  log also debugging messages, i.e. messages which show development and low-level I/O information.

Example

```
SSLLogLevel warn
```

SSLMutex Directive

Description:	Semaphore for internal mutual exclusion of operations
Syntax:	SSLMutex *type*
Default:	`SSLMutex none`
Context:	server config
Status:	Extension
Module:	mod_ssl

This configures the SSL engine's semaphore (aka. lock) which is used for mutual exclusion of operations which have to be done in a synchronized way between the pre-forked Apache server processes. This directive can only be used in the global server context because it's only useful to have one global mutex.

The following Mutex *types* are available:

- `none`

  This is the default where no Mutex is used at all. Use it at your own risk. But because currently the Mutex is mainly used for synchronizing write access to the SSL Session Cache you can live without it as long as you accept a sometimes garbled Session Cache. So it's not recommended to leave this the default. Instead configure a real Mutex.
- `file:/path/to/mutex`

  This is the portable and (under Unix) always provided Mutex variant where a physical (lock-)file is used as the Mutex. Always use a local disk filesystem for `/path/to/mutex` and never a file residing on a NFS- or AFS-filesystem. Note: Internally, the Process ID (PID) of the Apache parent process is automatically appended to `/path/to/mutex` to make it unique, so you don't have to worry about conflicts yourself. Notice that this type of mutex is not available under the Win32 environment. There you *have* to use the semaphore mutex.
- `sem`

  This is the most elegant but also most non-portable Mutex variant where a SysV IPC Semaphore (under Unix) and a Windows Mutex (under Win32) is used when possible. It is only available when the underlying platform supports it.

Example

```
SSLMutex file:/usr/local/apache/logs/ssl_mutex
```

SSLOptions Directive

Description:	Configure various SSL engine run-time options
Syntax:	SSLOptions [+|-]*option* ...
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Extension
Module:	mod_ssl

This directive can be used to control various run-time options on a per-directory basis. Normally, if multiple `SSLOptions` could apply to a directory, then the most specific one is taken completely; the options are not merged. However if *all* the options on the `SSLOptions` directive are preceded by a plus (`+`) or minus (`-`) 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.

The available *option*s are:

- `StdEnvVars`

  When this option is enabled, the standard set of SSL related CGI/SSI environment variables are created. This per default is disabled for performance reasons, because the information extraction step is a rather expensive operation. So one usually enables this option for CGI and SSI requests only.
- `CompatEnvVars`

  When this option is enabled, additional CGI/SSI environment variables are created for backward compatibility to other Apache SSL solutions. Look in the Compatibility chapter for details on the particular variables generated.
- `ExportCertData`

  When this option is enabled, additional CGI/SSI environment variables are created: `SSL_SERVER_CERT`, `SSL_CLIENT_CERT` and `SSL_CLIENT_CERT_CHAIN`*n* (with *n* = 0,1,2,..). These contain the PEM-encoded X.509 Certificates of server and client for the current HTTPS connection and can be used by CGI scripts for deeper Certificate checking. Additionally all other certificates of the client certificate chain are provided, too. This bloats up the environment a little bit which is why you have to use this option to enable it on demand.

- `FakeBasicAuth`

  When this option is enabled, the Subject Distinguished Name (DN) of the Client X509 Certificate is translated into a HTTP Basic Authorization username. This means that the standard Apache authentication methods can be used for access control. The user name is just the Subject of the Client's X509 Certificate (can be determined by running OpenSSL's `openssl x509` command: `openssl x509 -noout -subject -in` *certificate*`.crt`). Note that no password is obtained from the user. Every entry in the user file needs this password: ``xxj31ZMTZzkVA'', which is the DES-encrypted version of the word `password''. Those who live under MD5-based encryption (for instance under FreeBSD or BSD/OS, etc.) should use the following MD5 hash of the same word: ``$1$OXLyS...$Owx8s2/m9/gfkcRVXzgoE/''.
- `StrictRequire`

  This *forces* forbidden access when `SSLRequireSSL` or `SSLRequire` successfully decided that access should be forbidden. Usually the default is that in the case where a ``Satisfy any'' directive is used, and other access restrictions are passed, denial of access due to `SSLRequireSSL` or `SSLRequire` is overridden (because that's how the Apache `Satisfy` mechanism should work.) But for strict access restriction you can use `SSLRequireSSL` and/or `SSLRequire` in combination with an ``SSLOptions +StrictRequire''. Then an additional ``Satisfy Any'' has no chance once mod_ssl has decided to deny access.
- `OptRenegotiate`

  This enables optimized SSL connection renegotiation handling when SSL directives are used in per-directory context. By default a strict scheme is enabled where *every* per-directory reconfiguration of SSL parameters causes a *full* SSL renegotiation handshake. When this option is used mod_ssl tries to avoid unnecessary handshakes by doing more granular (but still safe) parameter checks. Nevertheless these granular checks sometimes maybe not what the user expects, so enable this on a per-directory basis only, please.

Example

```
SSLOptions +FakeBasicAuth -StrictRequire
<Files ~ "\.(cgi|shtml)$">
SSLOptions +StdEnvVars +CompatEnvVars -ExportCertData
<Files>
```

SSLPassPhraseDialog Directive

Description:	Type of pass phrase dialog for encrypted private keys
Syntax:	SSLPassPhraseDialog *type*
Default:	`SSLPassPhraseDialog builtin`
Context:	server config
Status:	Extension
Module:	mod_ssl

When Apache starts up it has to read the various Certificate (see `SSLCertificateFile`) and Private Key (see `SSLCertificateKeyFile`) files of the SSL-enabled virtual servers. Because for security reasons the Private Key files are usually encrypted, mod_ssl needs to query the administrator for a Pass Phrase in order to decrypt those files. This query can be done in two ways which can be configured by *type*:

- `builtin`

  This is the default where an interactive terminal dialog occurs at startup time just before Apache detaches from the terminal. Here the administrator has to manually enter the Pass Phrase for each encrypted Private Key file. Because a lot of SSL-enabled virtual hosts can be configured, the following reuse-scheme is used to minimize the dialog: When a Private Key file is encrypted, all known Pass Phrases (at the beginning there are none, of course) are tried. If one of those known Pass Phrases succeeds no dialog pops up for this particular Private Key file. If none succeeded, another Pass Phrase is queried on the terminal and remembered for the next round (where it perhaps can be reused).

  This scheme allows mod_ssl to be maximally flexible (because for N encrypted Private Key files you *can* use N different Pass Phrases - but then you have to enter all of them, of course) while minimizing the terminal dialog (i.e. when you use a single Pass Phrase for all N Private Key files this Pass Phrase is queried only once).
- `exec:/path/to/program`

  Here an external program is configured which is called at startup for each encrypted Private Key file. It is called with two arguments (the first is of the form ``servername:portnumber'', the second is either ``RSA'' or ``DSA''), which indicate for which server and algorithm it has to print the corresponding Pass Phrase to `stdout`. The intent is that this external program first runs security checks to make sure that the system is not compromised by an attacker, and only when these checks were passed successfully it provides the Pass Phrase.

  Both these security checks, and the way the Pass Phrase is determined, can be as complex as you like. Mod_ssl just defines the interface: an executable program which provides the Pass Phrase on `stdout`. Nothing more or less! So, if you're really paranoid about security, here is your interface. Anything else has to be left as an exercise to the administrator, because local security requirements are so different.

The reuse-algorithm above is used here, too. In other words: The external program is called only once per unique Pass Phrase.

Example:

```
SSLPassPhraseDialog exec:/usr/local/apache/sbin/pp-filter
```

SSLProtocol Directive

Description:	Configure usable SSL protocol flavors
Syntax:	SSLProtocol [+|-]*protocol* ...
Default:	`SSLProtocol all`
Context:	server config, virtual host
Override:	Options
Status:	Extension
Module:	mod_ssl

This directive can be used to control the SSL protocol flavors mod_ssl should use when establishing its server environment. Clients then can only connect with one of the provided protocols.

The available (case-insensitive) *protocol*s are:

- `SSLv2`

  This is the Secure Sockets Layer (SSL) protocol, version 2.0. It is the original SSL protocol as designed by Netscape Corporation.
- `SSLv3`

  This is the Secure Sockets Layer (SSL) protocol, version 3.0. It is the successor to SSLv2 and the currently (as of February 1999) de-facto standardized SSL protocol from Netscape Corporation. It's supported by almost all popular browsers.
- `TLSv1`

  This is the Transport Layer Security (TLS) protocol, version 1.0. It is the successor to SSLv3 and currently (as of February 1999) still under construction by the Internet Engineering Task Force (IETF). It's still not supported by any popular browsers.
- `All`

  This is a shortcut for ``` ``+SSLv2 +SSLv3 +TLSv1" ``` and a convinient way for enabling all protocols except one when used in combination with the minus sign on a protocol as the example above shows.

Example

```
# enable SSLv3 and TLSv1, but not SSLv2
SSLProtocol all -SSLv2
```

SSLRandomSeed Directive

Description:	Pseudo Random Number Generator (PRNG) seeding source
Syntax:	SSLRandomSeed *context source* [*bytes*]
Context:	server config
Status:	Extension
Module:	mod_ssl

This configures one or more sources for seeding the Pseudo Random Number Generator (PRNG) in OpenSSL at startup time (*context* is `startup`) and/or just before a new SSL connection is established (*context* is `connect`). This directive can only be used in the global server context because the PRNG is a global facility.

The following *source* variants are available:

- `builtin`

  This is the always available builtin seeding source. It's usage consumes minimum CPU cycles under runtime and hence can be always used without drawbacks. The source used for seeding the PRNG contains of the current time, the current process id and (when applicable) a randomly choosen 1KB extract of the inter-process scoreboard structure of Apache. The drawback is that this is not really a strong source and at startup time (where the scoreboard is still not available) this source just produces a few bytes of entropy. So you should always, at least for the startup, use an additional seeding source.

- `file:/path/to/source`

  This variant uses an external file `/path/to/source` as the source for seeding the PRNG. When *bytes* is specified, only the first *bytes* number of bytes of the file form the entropy (and *bytes* is given to `/path/to/source` as the first argument). When *bytes* is not specified the whole file forms the entropy (and `0` is given to `/path/to/source` as the first argument). Use this especially at startup time, for instance with an available `/dev/random` and/or `/dev/urandom` devices (which usually exist on modern Unix derivates like FreeBSD and Linux).

  But be careful: Usually `/dev/random` provides only as much entropy data as it actually has, i.e. when you request 512 bytes of entropy, but the device currently has only 100 bytes available two things can happen: On some platforms you receive only the 100 bytes while on other platforms the read blocks until enough bytes are available (which can take a long time). Here using an existing `/dev/urandom` is better, because it never blocks and actually gives the amount of requested data. The drawback is just that the quality of the received data may not be the best.

  On some platforms like FreeBSD one can even control how the entropy is actually generated, i.e. by which system interrupts. More details one can find under *rndcontrol(8)* on those platforms. Alternatively, when your system lacks such a random device, you can use tool like EGD (Entropy Gathering Daemon) and run it's client program with the `exec:/path/to/program/` variant (see below) or use `egd:/path/to/egd-socket` (see below).

- `exec:/path/to/program`

  This variant uses an external executable `/path/to/program` as the source for seeding the PRNG. When *bytes* is specified, only the first *bytes* number of bytes of its `stdout` contents form the entropy. When *bytes* is not specified, the entirety of the data produced on `stdout` form the entropy. Use this only at startup time when you need a very strong seeding with the help of an external program (for instance as in the example above with the `truerand` utility you can find in the mod_ssl distribution which is based on the AT&T *truerand* library). Using this in the connection context slows down the server too dramatically, of course. So usually you should avoid using external programs in that context.

- `egd:/path/to/egd-socket` (Unix only)

  This variant uses the Unix domain socket of the external Entropy Gathering Daemon (EGD) (see http://www.lothar.com/tech /crypto/) to seed the PRNG. Use this if no random device exists on your platform.

Example

```
SSLRandomSeed startup builtin
SSLRandomSeed startup file:/dev/random
SSLRandomSeed startup file:/dev/urandom 1024
SSLRandomSeed startup exec:/usr/local/bin/truerand 16
SSLRandomSeed connect builtin
SSLRandomSeed connect file:/dev/random
SSLRandomSeed connect file:/dev/urandom 1024
```

SSLRequire Directive

Description:	Allow access only when an arbitrarily complex boolean expression is true
Syntax:	SSLRequire *expression*
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive specifies a general access requirement which has to be fulfilled in order to allow access. It's a very powerful directive because the requirement specification is an arbitrarily complex boolean expression containing any number of access checks.

The *expression* must match the following syntax (given as a BNF grammar notation):

```
expr     ::= "true" | "false"
           | "!" expr
           | expr "&&" expr
           | expr "||" expr
           | "(" expr ")"
           | comp

comp     ::= word "==" word | word "eq" word
           | word "!=" word | word "ne" word
```

```
                      | word "<"  word | word "lt" word
                      | word "<=" word | word "le" word
                      | word ">"  word | word "gt" word
                      | word ">=" word | word "ge" word
                      | word "in" "{" wordlist "}"
                      | word "=~" regex
                      | word "!~" regex

        wordlist ::= word
                   | wordlist "," word

        word     ::= digit
                   | cstring
                   | variable
                   | function

        digit    ::= [0-9]+
        cstring  ::= "..."
        variable ::= "%{" varname "}"
        function ::= funcname "(" funcargs ")"
```

while for `varname` any variable from Table 3 can be used. Finally for `funcname` the following functions are available:

- `file(`*filename*`)`

  This function takes one string argument and expands to the contents of the file. This is especially useful for matching this contents against a regular expression, etc.

Notice that *expression* is first parsed into an internal machine representation and then evaluated in a second step. Actually, in Global and Per-Server Class context *expression* is parsed at startup time and at runtime only the machine representation is executed. For Per-Directory context this is different: here *expression* has to be parsed and immediately executed for every request.

Example

```
SSLRequire ( %{SSL_CIPHER} !~ m/^(EXP|NULL)-/ \
and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." \
and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"} \
and %{TIME_WDAY} >= 1 and %{TIME_WDAY} <= 5 \
and %{TIME_HOUR} >= 8 and %{TIME_HOUR} <= 20 ) \
or %{REMOTE_ADDR} =~ m/^192\.76\.162\.[0-9]+$/
```

Standard CGI/1.0 and Apache variables:

```
HTTP_USER_AGENT        PATH_INFO          AUTH_TYPE
HTTP_REFERER           QUERY_STRING       SERVER_SOFTWARE
HTTP_COOKIE            REMOTE_HOST        API_VERSION
HTTP_FORWARDED         REMOTE_IDENT       TIME_YEAR
HTTP_HOST              IS_SUBREQ          TIME_MON
HTTP_PROXY_CONNECTION  DOCUMENT_ROOT      TIME_DAY
HTTP_ACCEPT            SERVER_ADMIN       TIME_HOUR
HTTP:headername        SERVER_NAME        TIME_MIN
THE_REQUEST            SERVER_PORT        TIME_SEC
REQUEST_METHOD         SERVER_PROTOCOL    TIME_WDAY
REQUEST_SCHEME         REMOTE_ADDR        TIME
REQUEST_URI            REMOTE_USER        ENV:variablename
REQUEST_FILENAME
```

SSL-related variables:

```
HTTPS                  SSL_CLIENT_M_VERSION   SSL_SERVER_M_VERSION
                       SSL_CLIENT_M_SERIAL    SSL_SERVER_M_SERIAL
SSL_PROTOCOL           SSL_CLIENT_V_START     SSL_SERVER_V_START
SSL_SESSION_ID         SSL_CLIENT_V_END       SSL_SERVER_V_END
SSL_CIPHER             SSL_CLIENT_S_DN        SSL_SERVER_S_DN
SSL_CIPHER_EXPORT      SSL_CLIENT_S_DN_C      SSL_SERVER_S_DN_C
SSL_CIPHER_ALGKEYSIZE  SSL_CLIENT_S_DN_ST     SSL_SERVER_S_DN_ST
SSL_CIPHER_USEKEYSIZE  SSL_CLIENT_S_DN_L      SSL_SERVER_S_DN_L
SSL_VERSION_LIBRARY    SSL_CLIENT_S_DN_O      SSL_SERVER_S_DN_O
SSL_VERSION_INTERFACE  SSL_CLIENT_S_DN_OU     SSL_SERVER_S_DN_OU
                       SSL_CLIENT_S_DN_CN     SSL_SERVER_S_DN_CN
                       SSL_CLIENT_S_DN_T      SSL_SERVER_S_DN_T
                       SSL_CLIENT_S_DN_I      SSL_SERVER_S_DN_I
                       SSL_CLIENT_S_DN_G      SSL_SERVER_S_DN_G
                       SSL_CLIENT_S_DN_S      SSL_SERVER_S_DN_S
```

SSL_CLIENT_S_DN_D	SSL_SERVER_S_DN_D
SSL_CLIENT_S_DN_UID	SSL_SERVER_S_DN_UID
SSL_CLIENT_S_DN_Email	SSL_SERVER_S_DN_Email
SSL_CLIENT_I_DN	SSL_SERVER_I_DN
SSL_CLIENT_I_DN_C	SSL_SERVER_I_DN_C
SSL_CLIENT_I_DN_ST	SSL_SERVER_I_DN_ST
SSL_CLIENT_I_DN_L	SSL_SERVER_I_DN_L
SSL_CLIENT_I_DN_O	SSL_SERVER_I_DN_O
SSL_CLIENT_I_DN_OU	SSL_SERVER_I_DN_OU
SSL_CLIENT_I_DN_CN	SSL_SERVER_I_DN_CN
SSL_CLIENT_I_DN_T	SSL_SERVER_I_DN_T
SSL_CLIENT_I_DN_I	SSL_SERVER_I_DN_I
SSL_CLIENT_I_DN_G	SSL_SERVER_I_DN_G
SSL_CLIENT_I_DN_S	SSL_SERVER_I_DN_S
SSL_CLIENT_I_DN_D	SSL_SERVER_I_DN_D
SSL_CLIENT_I_DN_UID	SSL_SERVER_I_DN_UID
SSL_CLIENT_I_DN_Email	SSL_SERVER_I_DN_Email
SSL_CLIENT_A_SIG	SSL_SERVER_A_SIG
SSL_CLIENT_A_KEY	SSL_SERVER_A_KEY
SSL_CLIENT_CERT	SSL_SERVER_CERT
SSL_CLIENT_CERT_CHAIN**n**	
SSL_CLIENT_VERIFY	

Table 3: Available Variables for SSLRequire

SSLRequireSSL Directive

Description:	Deny access when SSL is not used for the HTTP request
Syntax:	SSLRequireSSL
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive forbids access unless HTTP over SSL (i.e. HTTPS) is enabled for the current connection. This is very handy inside the SSL-enabled virtual host or directories for defending against configuration errors that expose stuff that should be protected. When this directive is present all requests are denied which are not using SSL.

Example

```
SSLRequireSSL
```

SSLSessionCache Directive

Description:	Type of the global/inter-process SSL Session Cache
Syntax:	SSLSessionCache *type*
Default:	`SSLSessionCache none`
Context:	server config
Status:	Extension
Module:	mod_ssl

This configures the storage type of the global/inter-process SSL Session Cache. This cache is an optional facility which speeds up parallel request processing. For requests to the same server process (via HTTP keep-alive), OpenSSL already caches the SSL session information locally. But because modern clients request inlined images and other data via parallel requests (usually up to four parallel requests are common) those requests are served by *different* pre-forked server processes. Here an inter-process cache helps to avoid unneccessary session handshakes.

The following two storage *type*s are currently supported:

- `none`

  This is the default and just disables the global/inter-process Session Cache. There is no drawback in functionality, but a noticeable speed penalty can be observed.
- `dbm:/path/to/datafile`

This makes use of a DBM hashfile on the local disk to synchronize the local OpenSSL memory caches of the server processes. The slight increase in I/O on the server results in a visible request speedup for your clients, so this type of storage is generally recommended.

- `shm:/path/to/datafile[(`*size*`)]`

  This makes use of a high-performance hash table (approx. *size* bytes in size) inside a shared memory segment in RAM (established via `/path/to/datafile`) to synchronize the local OpenSSL memory caches of the server processes. This storage type is not available on all platforms. See the mod_ssl `INSTALL` document for details on how to build Apache+EAPI with shared memory support.

Examples

```
SSLSessionCache dbm:/usr/local/apache/logs/ssl_gcache_data
SSLSessionCache shm:/usr/local/apache/logs/ssl_gcache_data(512000)
```

SSLSessionCacheTimeout Directive

Description:	Number of seconds before an SSL session expires in the Session Cache
Syntax:	SSLSessionCacheTimeout *seconds*
Default:	`SSLSessionCacheTimeout 300`
Context:	server config, virtual host
Status:	Extension
Module:	mod_ssl

This directive sets the timeout in seconds for the information stored in the global/inter-process SSL Session Cache and the OpenSSL internal memory cache. It can be set as low as 15 for testing, but should be set to higher values like 300 in real life.

Example

```
SSLSessionCacheTimeout 600
```

SSLVerifyClient Directive

Description:	Type of Client Certificate verification
Syntax:	SSLVerifyClient *level*
Default:	`SSLVerifyClient none`
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive sets the Certificate verification level for the Client Authentication. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the client authentication process used in the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotation with the reconfigured client verification level after the HTTP request was read but before the HTTP response is sent.

The following levels are available for *level*:

- **none**: no client Certificate is required at all
- **optional**: the client *may* present a valid Certificate
- **require**: the client *has to* present a valid Certificate
- **optional_no_ca**: the client may present a valid Certificate
  but it need not to be (successfully) verifiable.

In practice only levels **none** and **require** are really interesting, because level **optional** doesn't work with all browsers and level **optional_no_ca** is actually against the idea of authentication (but can be used to establish SSL test pages, etc.)

Example

```
SSLVerifyClient require
```

SSLVerifyDepth Directive

Description:	Maximum depth of CA Certificates in Client Certificate verification
Syntax:	SSLVerifyDepth *number*
Default:	`SSLVerifyDepth 1`
Context:	server config, virtual host, directory, .htaccess
Override:	AuthConfig
Status:	Extension
Module:	mod_ssl

This directive sets how deeply mod_ssl should verify before deciding that the clients don't have a valid certificate. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the client authentication process used in the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotation with the reconfigured client verification depth after the HTTP request was read but before the HTTP response is sent.

The depth actually is the maximum number of intermediate certificate issuers, i.e. the number of CA certificates which are max allowed to be followed while verifying the client certificate. A depth of 0 means that self-signed client certificates are accepted only, the default depth of 1 means the client certificate can be self-signed or has to be signed by a CA which is directly known to the server (i.e. the CA's certificate is under `SSLCACertificatePath`), etc.

Example

```
SSLVerifyDepth 10
```

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Apache Module mod_status

Description:	Provides information on server activity and performance
Status:	Base
Module Identifier:	status_module

Summary

Warning: This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

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 (*)

A compile-time option must be used to display the details marked "(*)" as the instrumentation required for obtaining these statistics does not exist within standard Apache.

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

Description:	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.
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 setting applies to the entire server, and cannot be enabled or disabled on a virtualhost-by-virtualhost basis.

Apache HTTP Server Version 2.0

Apache Module mod_suexec

Description:	Allows CGI scripts to run as a specified user and Group
Status:	Extension
Module Identifier:	suexec_module
Compatibility:	Available in Apache 2.0 and later

Summary

This module allows CGI scripts to run as a specified user and Group.

Directives

- SuexecUserGroup

SuexecUserGroup Directive

Description:	
Syntax:	SuexecUserGroup *User Group*
Default:	`None`
Context:	server config, virtual host
Status:	Extension
Module:	mod_suexec
Compatibility:	SuexecUserGroup is only available in 2.0 and later.

The `SuexecUserGroup` directive allows you to specify a user and group for CGI programs to run as. Non-CGI requests are still processes with the user specified in the User directive. This directive replaces using the User and Group directives inside of VirtualHosts.

Apache HTTP Server Version 2.0

Apache Module mod_unique_id

Description:	Provides an environment variable with a unique identifier for each request
Status:	Extension
Module Identifier:	unique_id_module

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 provides no directives.

Theory

First a brief recap of how the Apache server works on Unix machines. This feature currently isn't supported on Windows NT. 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_ID`s 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_ID`s. 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 we believe is a relatively portable solution to this problem. It can be extended to multithreaded systems like Windows NT, and can grow with future needs. 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).

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_userdir

Description:	Provides for user-specific directories
Status:	Base
Module Identifier:	userdir_module

Summary

Directives

- UserDir

UserDir Directive

Description:	Sets the directory from which to serve files when requests for a particular user are received, denoted by requests containing ~username, such as http://server.example.com/~bob/
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 directive used	Translated path
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 directive used	Translated path

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.

Additional examples:

To allow a few users to have `UserDir` directories, but not anyone else, use the following:

```
UserDir disabled
UserDir enabled user1 user2 user3
```

To allow most users to have `UserDir` directories, but deny this to a few, use the following:

```
UserDir enabled
UserDir disabled user4 user5 user6
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Module mod_usertrack

Description:	This module uses cookies to provide for a *clickstream* log of user activity on a site.
Status:	Extension
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

Description:	controls the setting of the domain to which the tracking cookie applies.
Syntax:	CookieDomain *domain*
Default:	`None`
Context:	server config, virtual host, directory, .htaccess
Status:	Extension
Module:	mod_usertrack

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

Description:	
Syntax:	CookieExpires *expiry-period*
Default:	
Context:	server config, virtual host, directory, .htaccess
Override:	
Status:	Extension
Module:	mod_usertrack
Compatibility:	In 1.3.20 and earlier, not usable in directory and .htaccess

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

Description:	
Syntax:	CookieName *token*
Default:	`Apache`
Context:	server config, virtual host, directory, .htaccess
Status:	Extension
Module:	mod_usertrack

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

Description:	Controls the format of the cookie header field
Syntax:	CookieStyle *Netscape|Cookie|Cookie2|RFC2109|RFC2965*
Default:	
Context:	server config, virtual host, directory, .htaccess
Status:	Extension
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

Description:	
Syntax:	CookieTracking on|off
Default:	
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Extension
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 2.0

Apache HTTP Server Version 2.0

Apache Module mod_vhost_alias

Description:	Provides for dynamically configured mass virtual hosting
Status:	Extension
Module Identifier:	vhost_alias_module

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`	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.isp.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.isp.com/directory/file.html` will be satisfied by the file `/usr/local/apache/vhosts/example.isp/directory/file.html`.

The `LogFormat` directives `%V` and `%A` are useful in conjunction with this module.

VirtualDocumentRoot Directive

Description:	Dynamically configure the location of the document root for a given virtual host
Syntax:	VirtualDocumentRoot *interpolated-directory*
Default:	`none`
Context:	server config, virtual host
Override:	
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 `VirtaulDocumentRoot` is turned off. This directive cannot be used in the same context as `VirtualDocumentRootIP`.

VirtualDocumentRootIP Directive

Description:	Dynamically configure the location of the document root for a given virtual host
Syntax:	VirtualDocumentRootIP *interpolated-directory*
Default:	`none`
Context:	server config, virtual host
Override:	
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

Description:	Dynamically configure the location of the CGI directory for a given virtual host
Syntax:	VirtualScriptAlias *interpolated-directory*
Default:	`none`
Context:	server config, virtual host
Override:	
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

Description:	Dynamically configure the location of the cgi directory for a given virtual host
Syntax:	VirtualScriptAliasIP *interpolated-directory*
Default:	`none`
Context:	server config, virtual host
Override:	
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 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Developer Documentation for Apache-2.0

Many of the documents on these Developer pages are lifted from Apache 1.3's documentation. While they are all being updated to Apache 2.0, they are in different stages of progress. Please be patient, and point out any discrepancies or errors on the developer/ pages directly to the dev@httpd.apache.org mailing list.

Topics

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Warning: This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

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.

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 `ScriptAlias`ed URIs), the type-checking phase (any `ScriptAlias`ed request is typed as a CGI script).

The module needs to maintain some per (virtual) server information, namely, the `ScriptAlias`es 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 `ScriptAlias`es 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 `GET`s 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_pfopen`, which also arranges for the underlying file descriptor to be closed before any child processes, such as for CGI scripts, are `exec`ed), 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 `char`s). 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`ed 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 `free`d, 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_pcalloc` 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 themself, 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_pfopen`, 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_pfopen (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_pfopen`, 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_pfopen` 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 `Alias`es and `Redirect`s 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 `srm.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 `table`s 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 `srm.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_palloc`s the structure above, and a creates a couple of `table`s 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 2.0

Apache HTTP Server Version 2.0

Debugging Memory Allocation in APR

The allocation mechanism's within APR have a number of debugging modes that can be used to assist in finding memory problems. This document describes the modes available and gives instructions on activating them.

Allocation Debugging

ALLOC_DEBUG

Debugging support: Define this to enable code which helps detect re-use of freed memory and other such nonsense.

The theory is simple. The FILL_BYTE (0xa5) is written over all malloc'd memory as we receive it, and is written over everything that we free up during a clear_pool. We check that blocks on the free list always have the FILL_BYTE in them, and we check during palloc() that the bytes still have FILL_BYTE in them. If you ever see garbage URLs or whatnot containing lots of 0xa5s then you know something used data that's been freed or uninitialized.

Malloc Support

ALLOC_USE_MALLOC

If defined all allocations will be done with malloc and free()d appropriately at the end.

This is intended to be used with something like Electric Fence or Purify to help detect memory problems. Note that if you're using efence then you should also add in ALLOC_DEBUG. But don't add in ALLOC_DEBUG if you're using Purify because ALLOC_DEBUG would hide all the uninitialized read errors that Purify can diagnose.

Pool Debugging

POOL_DEBUG

This is intended to detect cases where the wrong pool is used when assigning data to an object in another pool.

In particular, it causes the table_{set,add,merge}n routines to check that their arguments are safe for the apr_table_t they're being placed in. It currently only works with the unix multiprocess model, but could be extended to others.

Table Debugging

MAKE_TABLE_PROFILE

Provide diagnostic information about make_table() calls which are possibly too small.

This requires a recent gcc which supports __builtin_return_address(). The error_log output will be a message such as:

```
table_push: apr_table_t created by 0x804d874 hit limit of 10
```

Use "***l *0x804d874***" to find the source that corresponds to. It indicates that a apr_table_t allocated by a call at that address has possibly too small an initial apr_table_t size guess.

Allocation Statistics

ALLOC_STATS

Provide some statistics on the cost of allocations.

This requires a bit of an understanding of how alloc.c works.

Allowable Combinations

Not all the options outlined above can be activated at the same time. the following table gives more information.

Option 1	**ALLOC DEBUG**	**ALLOC USE MALLOC**	**POOL DEBUG**	**MAKE TABLE PROFILE**	**ALLOC STATS**
ALLOC_DEBUG	🟥	No	Yes	Yes	Yes
ALLOC_USE MALLOC	No	🟥	No	No	No
POOL_DEBUG	Yes	No	🟥	Yes	Yes
MAKE_TABLE PROFILE	Yes	No	Yes	🟥	Yes
ALLOC_STATS	Yes	No	Yes	Yes	🟥

Additionally the debugging options are not suitable for multi-threaded versions of the server. When trying to debug with these options the server should be started in single process mode.

Activating Debugging Options

The various options for debugging memory are now enabled in the apr_general.h header file in APR. The various options are enabled by uncommenting the define for the option you wish to use. The section of the code currently looks like this *(contained in srclib/apr/include/apr_pools.h)*

```
/*
#define ALLOC_DEBUG
#define POOL_DEBUG
#define ALLOC_USE_MALLOC
#define MAKE_TABLE_PROFILE
#define ALLOC_STATS
*/

typedef struct ap_pool_t {
    union block_hdr *first;
    union block_hdr *last;
    struct cleanup *cleanups;
    struct process_chain *subprocesses;
    struct ap_pool_t *sub_pools;
    struct ap_pool_t *sub_next;
    struct ap_pool_t *sub_prev;
    struct ap_pool_t *parent;
    char *free_first_avail;
#ifdef ALLOC_USE_MALLOC
    void *allocation_list;
#endif
#ifdef POOL_DEBUG
    struct ap_pool_t *joined;
#endif
    int (*apr_abort)(int retcode);
    struct datastruct *prog_data;
}ap_pool_t;
```

To enable allocation debugging simply move the #define ALLOC_DEBUG above the start of the comments block and rebuild the server.

NB. In order to use the various options the server MUST be rebuilt after editing the header file.

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Documentating Apache 2.0

Apache 2.0 uses DoxyGen to document the API's and global variables in the the code. This will explain the basics of how to document using DoxyGen.

To start a documentation block, use /**
To end a documentation block, use */

In the middle of the block, there are multiple tags we can use:

```
    Description of this functions purpose
    @param parameter_name description
```

```
The deffunc is not always necessary.  DoxyGen does not have a full parser
   in it, so any prototype that use a macro in the return type declaration
   is too complex for scandoc.  Those functions require a deffunc.
```

```
An example (using &gt; rather than >):
```

```
/**
 * return the final element of the pathname
 * @param pathname The path to get the final element of
 * @return the final element of the path
 * @tip Examples:
 * <pre>
 *                 "/foo/bar/gum"   -&gt; "gum"
 *                 "/foo/bar/gum/"  -&gt; ""
 *                 "gum"            -&gt; "gum"
 *                 "wi\\n32\\stuff" -&gt; "stuff"
 * </pre>
 * @deffunc const char * ap_filename_of_pathname(const char *pathname)
 */
```

```
At the top of the header file, always include:
```

```
/**
 * @package Name of library header
 */
```

```
ScanDoc uses a new html file for each package.  The html files are named
   {Name_of_library_header}.html, so try to be concise with your names.
```

Apache HTTP Server Version 2.0

Apache HTTP Server Version 2.0

Apache Hook Functions

In general, a hook function is one that Apache will call at some point during the processing of a request. Modules can provide functions that are called, and specify when they get called in comparison to other modules.

Creating a hook function

In order to create a new hook, four things need to be done:

Declare the hook function

Use the AP_DECLARE_HOOK macro, which needs to be given the return type of the hook function, the name of the hook, and the arguments. For example, if the hook returns an `int` and takes a `request_rec *` and an `int` and is called "do_something", then declare it like this:

```
AP_DECLARE_HOOK(int,do_something,(request_rec *r,int n))
```

This should go in a header which modules will include if they want to use the hook.

Create the hook structure

Each source file that exports a hook has a private structure which is used to record the module functions that use the hook. This is declared as follows:

```
APR_HOOK_STRUCT(
            APR_HOOK_LINK(do_something)
            ...
           )
```

Implement the hook caller

The source file that exports the hook has to implement a function that will call the hook. There are currently three possible ways to do this. In all cases, the calling function is called `ap_run_`*`hookname`*`()`.

Void hooks

If the return value of a hook is `void`, then all the hooks are called, and the caller is implemented like this:

```
AP_IMPLEMENT_HOOK_VOID(do_something,(request_rec *r,int n),(r,n))
```

The second and third arguments are the dummy argument declaration and the dummy arguments as they will be used when calling the hook. In other words, this macro expands to something like this:

```
void ap_run_do_something(request_rec *r,int n)
{
    ...
    do_something(r,n);
}
```

Hooks that return a value

If the hook returns a value, then it can either be run until the first hook that does something interesting, like so:

```
AP_IMPLEMENT_HOOK_RUN_FIRST(int,do_something,(request_rec *r,int n),(r,n),DECLINED)
```

The first hook that *doesn't* return `DECLINED` stops the loop and its return value is returned from the hook caller. Note that `DECLINED` is the tradition Apache hook return meaning "I didn't do anything", but it can be whatever suits you.

Alternatively, all hooks can be run until an error occurs. This boils down to permitting *two* return values, one of which means "I did something, and it was OK" and the other meaning "I did nothing". The first function that returns a value other than one of those two stops the loop, and its return is the return value. Declare these like so:

```
AP_IMPLEMENT_HOOK_RUN_ALL(int,do_something,(request_rec *r,int n),(r,n),OK,DECLINED)
```

Again, `OK` and `DECLINED` are the traditional values. You can use what you want.

Call the hook callers

At appropriate moments in the code, call the hook caller, like so:

```
    int n,ret;
    request_rec *r;

    ret=ap_run_do_something(r,n);
```

Hooking the hook

A module that wants a hook to be called needs to do two things.

Implement the hook function

Include the appropriate header, and define a static function of the correct type:

```
static int my_something_doer(request_rec *r,int n)
{
    ...
    return OK;
}
```

Add a hook registering function

During initialisation, Apache will call each modules hook registering function, which is included in the module structure:

```
static void my_register_hooks()
{
    ap_hook_do_something(my_something_doer,NULL,NULL,HOOK_MIDDLE);
}

mode MODULE_VAR_EXPORT my_module =
{
    ...
    my_register_hooks       /* register hooks */
};
```

Controlling hook calling order

In the example above, we didn't use the three arguments in the hook registration function that control calling order. There are two mechanisms for doing this. The first, rather crude, method, allows us to specify roughly where the hook is run relative to other modules. The final argument control this. There are three possible values:

```
HOOK_FIRST
HOOK_MIDDLE
HOOK_LAST
```

All modules using any particular value may be run in any order relative to each other, but, of course, all modules using `HOOK_FIRST` will be run before `HOOK_MIDDLE` which are before `HOOK_LAST`. Modules that don't care when they are run should use `HOOK_MIDDLE`. *(I spaced these out so people could do stuff like `HOOK_FIRST-2` to get in slightly earlier, but is this wise? - Ben)*

Note that there are two more values, `HOOK_REALLY_FIRST` and `HOOK_REALLY_LAST`. These should only be used by the hook exporter.

The other method allows finer control. When a module knows that it must be run before (or after) some other modules, it can specify them by name. The second (third) argument is a NULL-terminated array of strings consisting of the names of modules that must be run before (after) the current module. For example, suppose we want "mod_xyz.c" and "mod_abc.c" to run before we do, then we'd hook as follows:

```
static void register_hooks()
{
    static const char * const aszPre[]={ "mod_xyz.c", "mod_abc.c", NULL };

    ap_hook_do_something(my_something_doer,aszPre,NULL,HOOK_MIDDLE);
}
```

Note that the sort used to achieve this is stable, so ordering set by `HOOK_`*`ORDER`* is preserved, as far as is possible.

Ben Laurie, 15th August 1999

Apache HTTP Server Version 2.0

INDEX HOME

Apache Layered I/O

Layered I/O has been the holy grail of Apache module writers for years. With Apache 2.0, module writers can finally take advantage of layered I/O in their modules.

In all previous versions of Apache, only one handler was allowed to modify the data stream that was sent to the client. With Apache 2.0, one module can modify the data and then specify that other modules can modify the data if they would like.

Taking advantage of layered I/O

In order to make a module use layered I/O, there are some modifications needed. A new return value has been added for modules, RERUN_HANDLERS. When a handler returns this value, the core searches through the list of handlers looking for another module that wants to try the request.

When a module returns RERUN_HANDLERS, it must modify two fields of the request_rec, the handler and content_type fields. Most modules will set the handler field to NULL, and allow the core to choose the which module gets run next. If these two fields are not modified, then the server will loop forever calling the same module's handler.

Most modules should not write out to the network if they want to take advantage of layered I/O. Two BUFF structures have been added to the request_rec, one for input and one for output. The module should read and write to these BUFFs. The module will also have to setup the input field for the next module in the list. A new function has been added, ap_setup_input, which all modules should call before they do any reading to get data to modify. This function checks to determine if the previous module set the input field, if so, that input is used, if not the file is opened and that data source is used. The output field is used basically the same way. The module must set this field before they call ap_r* in order to take advantage of layered I/O. If this field is not set, ap_r* will write directly to the client. Usually at the end of a handler, the input (for the next module) will be the read side of a pipe, and the output will be the write side of the same pipe.

An Example of Layered I/O.

This example is the most basic layered I/O example possible. It is basically CGIs generated by mod_cgi and sent to the network via http_core.

mod_cgi executes the cgi script, and then sets request_rec->input to the output pipe of the CGI. It then NULLs out request_rec->handler, and sets request_rec->content_type to whatever the CGI writes out (in this case, text/html). Finally, mod_cgi returns RERUN_HANDLERS.

ap_invoke_handlers() then loops back to the top of the handler list and searches for a handler that can deal with this content_type. In this case the correct module is the default_handler from http_core.

When default handler starts, it calls ap_setup_input, which has found a valid request_rec->input, so that is used for all inputs. The output field in the request_rec is NULL, so when default_handler calls an output primitive it gets sent out over the network.

Ryan Bloom, 25th March 2000

Apache HTTP Server Version 2.0

From Apache 1.3 to Apache 2.0 Modules

This is a first attempt at writing the lessons I learned when trying to convert the mod_mmap_static module to Apache 2.0. It's by no means definitive and probably won't even be correct in some ways, but it's a start.

The easier changes...

Cleanup Routines

These now need to be of type apr_status_t and return a value of that type. Normally the return value will be APR_SUCCESS unless there is some need to signal an error in the cleanup. Be aware that even though you signal an error not all code yet checks and acts upon the error.

Initialisation Routines

These should now be renamed to better signify where they sit in the overall process. So the name gets a small change from mmap_init to mmap_post_config. The arguments passed have undergone a radical change and now look like

- apr_pool_t *p,
- apr_pool_t *plog,
- apr_pool_t *ptemp,
- server_rec *s

Data Types

A lot of the data types have been moved into the APR. This means that some have had a name change, such as the one shown above. The following is a brief list of some of the changes that you are likely to have to make.

- pool becomes apr_pool_t
- table becomes apr_table_t

The *messier* changes...

Register Hooks

The new architecture uses a series of hooks to provide for calling your functions. These you'll need to add to your module by way of a new function, static void register_hooks(void). The function is really reasonably straightforward once you understand what needs to be done. Each function that needs calling at some stage in the processing of a request needs to be registered, handlers do not. There are a number of phases where functions can be added, and for each you can specify with a high degree of control the relative order that the function will be called in.

This is the code that was added to mod_mmap_static:

```
static void register_hooks(void)
{
    static const char * const aszPre[]={ "http_core.c",NULL };
    ap_hook_post_config(mmap_post_config,NULL,NULL,HOOK_MIDDLE);
    ap_hook_translate_name(mmap_static_xlat,aszPre,NULL,HOOK_LAST);
};
```

This registers 2 functions that need to be called, one in the post_config stage (virtually every module will need this one) and one for the translate_name phase. note that while there are different function names the format of each is identical. So what is the format?

ap_hook_[phase_name](function_name, predecessors, successors, position);

There are 3 hook positions defined...

- HOOK_FIRST
- HOOK_MIDDLE
- HOOK_LAST

To define the position you use the position and then modify it with the predecessors and successors. each of the modifiers can be a list of functions that should be called, either before the function is run (predecessors) or after the function has run (successors).

In the mod_mmap_static case I didn't care about the post_config stage, but the mmap_static_xlat MUST be called after the core module had done it's name translation, hence the use of the aszPre to define a modifier to the position HOOK_LAST.

Module Definition

There are now a lot fewer stages to worry about when creating your module definition. The old defintion looked like

```
module MODULE_VAR_EXPORT [module_name]_module =
{
    STANDARD_MODULE_STUFF,
    /* initializer */
    /* dir config creater */
    /* dir merger --- default is to override */
    /* server config */
    /* merge server config */
    /* command handlers */
    /* handlers */
    /* filename translation */
    /* check_user_id */
    /* check auth */
    /* check access */
    /* type_checker */
    /* fixups */
    /* logger */
    /* header parser */
    /* child_init */
    /* child_exit */
    /* post read-request */
};
```

The new structure is a great deal simpler...

```
module MODULE_VAR_EXPORT [module_name]_module =
{
    STANDARD20_MODULE_STUFF,
    /* create per-directory config structures */
    /* merge per-directory config structures  */
    /* create per-server config structures    */
    /* merge per-server config structures     */
    /* command handlers */
    /* handlers */
    /* register hooks */
 };
```

Some of these read directly across, some don't. I'll try to summarise what should be done below.

The stages that read directly across :

- /* dir config creater */ ==> /* create per-directory config structures */

- /* server config */ ==> /* create per-server config structures */
- /* dir merger */ ==> /* merge per-directory config structures */
- /* merge server config */ ==> /* merge per-server config structures */
- /* command table */ ==> /* command apr_table_t */
- /* handlers */ ==> /* handlers */

The remainder of the old functions should be registered as hooks. There are the following hook stages defined so far...

- ap_hook_post_config *(this is where the old _init routines get registered)*
- ap_hook_http_method *(retrieve the http method from a request. (legacy))*
- ap_hook_open_logs *(open any specified logs)*
- ap_hook_auth_checker *(check if the resource requires authorization)*
- ap_hook_access_checker *(check for module-specific restrictions)*
- ap_hook_check_user_id *(check the user-id and password)*
- ap_hook_default_port *(retrieve the default port for the server)*
- ap_hook_pre_connection *(do any setup required just before processing, but after accepting)*
- ap_hook_process_connection *(run the correct protocol)*
- ap_hook_child_init *(call as soon as the child is started)*
- ap_hook_create_request *(??)*
- ap_hook_fixups *(last chance to modify things before generating content)*
- ap_hook_handler *(generate the content)*
- ap_hook_header_parser *(let's modules look at the headers, not used by most modules, because they use post_read_request for this.)*
- ap_hook_insert_filter *(to insert filters into the filter chain)*
- ap_hook_log_transaction *(log information about the request)*
- ap_hook_optional_fn_retrieve *(retrieve any functions registered as optional)*
- ap_hook_post_read_request *(called after reading the request, before any other phase)*
- ap_hook_quick_handler *(??)*
- ap_hook_translate_name *(translate the URI into a filename)*
- ap_hook_type_checker *(determine and/or set the doc type)*

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

Request Processing in Apache 2.0

Warning - this is a first (fast) draft that needs further revision!

Several changes in Apache 2.0 affect the internal request processing mechanics. Module authors need to be aware of these changes so they may take advantage of the optimizations and security enhancements.

The first major change is to the subrequest and redirect mechanisms. There were a number of different code paths in Apache 1.3 to attempt to optimize subrequest or redirect behavior. As patches were introduced to 2.0, these optimizations (and the server behavior) were quickly broken due to this duplication of code. All duplicate code has been folded back into `ap_process_internal_request()` to prevent the code from falling out of sync again.

This means that much of the existing code was 'unoptimized'. It is the Apache HTTP Project's first goal to create a robust and correct implementation of the HTTP server RFC. Additional goals include security, scalability and optimization. New methods were sought to optimize the server (beyond the performance of Apache 1.3) without introducing fragile or insecure code.

The Request Processing Cycle

All requests pass through `ap_process_request_internal()` in request.c, including subrequests and redirects. If a module doesn't pass generated requests through this code, the author is cautioned that the module may be broken by future changes to request processing.

To streamline requests, the module author can take advantage of the hooks offered to drop out of the request cycle early, or to bypass core Apache hooks which are irrelevant (and costly in terms of CPU.)

The Request Parsing Phase

Unescapes the URL

The request's parsed_uri path is unescaped, once and only once, at the beginning of internal request processing.

This step is bypassed if the proxyreq flag is set, or the parsed_uri.path element is unset. The module has no further control of this one-time unescape operation, either failing to unescape or multiply unescaping the URL leads to security reprecussions.

Strips Parent and This Elements from the URI

All `/../` and `/./` elements are removed by `ap_getparents()`. This helps to ensure the path is (nearly) absolute before the request processing continues.

This step cannot be bypassed.

Initial URI Location Walk

Every request is subject to an `ap_location_walk()` call. This ensures that <Location > sections are consistently enforced for all requests. If the request is an internal redirect or a sub-request, it may borrow some or all of the processing from the previous or parent request's ap_location_walk, so this step is generally very efficient after processing the main request.

Hook: translate_name

Modules can determine the file name, or alter the given URI in this step. For example, mod_vhost_alias will translate the URI's path into the configured virtual host, mod_alias will translate the path to an alias path, and if the request falls back on the core, the DocumentRoot is prepended to the request resource.

If all modules DECLINE this phase, an error 500 is returned to the browser, and a "couldn't translate name" error is logged automatically.

Hook: map_to_storage

After the file or correct URI was determined, the appropriate per-dir configurations are merged together. For example, mod_proxy compares and merges the appropriate <Proxy > sections. If the URI is nothing more than a local (non-proxy) TRACE request, the core handles the request and returns DONE. If no module answers this hook with OK or DONE, the core will run the request filename against the <Directory > and <Files > sections. If the request 'filename' isn't an absolute, legal filename, a note is set for later termination.

Initial URI Location Walk

Every request is hardened by a second `ap_location_walk()` call. This reassures that a translated request is still subjected to the configured <Location > sections. The request again borrows some or all of the processing from it's previous location_walk above, so this step is almost always very efficient unless the translated URI mapped to a substantially different path or Virtual Host.

Hook: header_parser

The main request then parses the client's headers. This prepares the remaining request processing steps to better serve the client's request.

The Security Phase

Needs Documentation. Code is;

```
    switch (ap_satisfies(r)) {
    case SATISFY_ALL:
    case SATISFY_NOSPEC:
        if ((access_status = ap_run_access_checker(r)) != 0) {
            return decl_die(access_status, "check access", r);
        }
        if (ap_some_auth_required(r)) {
            if (((access_status = ap_run_check_user_id(r)) != 0) || !ap_auth_type(r)) {
                return decl_die(access_status, ap_auth_type(r)
                    ? "check user.  No user file?"
                    : "perform authentication. AuthType not set!", r);
            }
            if (((access_status = ap_run_auth_checker(r)) != 0) || !ap_auth_type(r)) {
                return decl_die(access_status, ap_auth_type(r)
                    ? "check access.  No groups file?"
                    : "perform authentication. AuthType not set!", r);
            }
        }
        break;
    case SATISFY_ANY:
        if (((access_status = ap_run_access_checker(r)) != 0) || !ap_auth_type(r)) {
            if (!ap_some_auth_required(r)) {
                return decl_die(access_status, ap_auth_type(r)
                    ? "check access"
                    : "perform authentication. AuthType not set!", r);
            }
            if (((access_status = ap_run_check_user_id(r)) != 0) || !ap_auth_type(r)) {
                return decl_die(access_status, ap_auth_type(r)
                    ? "check user.  No user file?"
                    : "perform authentication. AuthType not set!", r);
            }
            if (((access_status = ap_run_auth_checker(r)) != 0) || !ap_auth_type(r)) {
                return decl_die(access_status, ap_auth_type(r)
                    ? "check access.  No groups file?"
                    : "perform authentication. AuthType not set!", r);
            }
            }
        break;
    }
```

The Preparation Phase

Hook: type_checker

The modules have an opportunity to test the URI or filename against the target resource, and set mime information for the request. Both mod_mime and mod_mime_magic use this phase to compare the file name or contents against the administrator's configuration and set the content type, language, character set and request handler. Some modules may set up their filters or other request handling parameters at this time.

If all modules DECLINE this phase, an error 500 is returned to the browser, and a "couldn't find types" error is logged automatically.

Hook: fixups

Many modules are 'trounced' by some phase above. The fixups phase is used by modules to 'reassert' their ownership or force the request's fields to their appropriate values. It isn't always the cleanest mechanism, but occasionally it's the only option.

Hook: insert_filter

Modules that transform the content in some way can insert their values and override existing filters, such that if the user configured a more advanced filter out-of-order, then the module can move it's order as need be.

The Handler Phase

This phase is ***not*** part of the processing in `ap_process_request_internal()`. Many modules prepare one or more subrequests prior to creating any content at all. After the core, or a module calls `ap_process_request_internal()` it then calls `ap_invoke_handler()` to generate the request.

Hook: handler

The module finally has a chance to serve the request in it's handler hook. Note that not every prepared request is sent to the handler hook. Many modules, such as mod_autoindex, will create subrequests for a given URI, and then never serve the subrequest, but simply lists it for the user. Remember not to put required teardown from the hooks above into this module, but register pool cleanups against the request pool to free resources as required.

Apache HTTP Server Version 2.0

INDEX HOME

Apache HTTP Server Version 2.0

How filters work in Apache 2.0

Warning - this is a cut 'n paste job from an email: <022501c1c529$f63a9550$7f00000a@KOJ>

```
There are three basic filter types (each of these is actually broken
down into two categories, but that comes later).

CONNECTION:  Filters of this type are valid for the lifetime of this
             connection. (AP_FTYPE_CONNECTION, AP_FTYPE_NETWORK)

PROTOCOL:    Filters of this type are valid for the lifetime of this
             request from the point of view of the client, this means
             that the request is valid from the time that the request
             is sent until the time that the response is received.
             (AP_FTYPE_PROTOCOL, AP_FTYPE_TRANSCODE)

RESOURCE:    Filters of this type are valid for the time that this
             content is used to satisfy a request.  For simple
             requests, this is identical to PROTOCOL, but internal redirects
             and sub-requests can change the content without ending
             the request. (AP_FTYPE_RESOURCE, AP_FTYPE_CONTENT_SET)

It is important to make the distinction between a protocol and a
resource filter.  A resource filter is tied to a specific resource, it
may also be tied to header information, but the main binding is to a
resource.  If you are writing a filter and you want to know if it is
resource or protocol, the correct question to ask is:  "Can this filter
be removed if the request is redirected to a different resource?"  If
the answer is yes, then it is a resource filter.  If it is no, then it
is most likely a protocol or connection filter.  I won't go into
connection filters, because they seem to be well understood.

With this definition, a few examples might help:
Byterange:  We have coded it to be inserted for all
requests, and it is removed if not used.  Because this filter is active
at the beginning of all requests, it can not be removed if it is
redirected, so this is a protocol filter.

http_header:  This filter actually writes the headers to the
network.  This is obviously a required filter (except in the asis case
which is special and will be dealt with below) and so it is a protocol
filter.

Deflate:  The administrator configures this filter based on
which file has been requested.  If we do an internal redirect from an
autoindex page to an index.html page, the deflate filter may be added or
removed based on config, so this is a resource filter.

The further breakdown of each category into two more filter types is
strictly for ordering.  We could remove it, and only allow for one
filter type, but the order would tend to be wrong, and we would need to
hack things to make it work.  Currently, the RESOURCE filters only have
one filter type, but that should change.

How are filters inserted?
This is actually rather simple in theory, but the code is
```

complex. First of all, it is important that everybody realize that there are three filter lists for each request, but they are all concatenated together. So, the first list is r->output_filters, then r->proto_output_filters, and finally r->connection->output_filters. These correspond to the RESOURCE, PROTOCOL, and CONNECTION filters respectively. The problem previously, was that we used a singly linked list to create the filter stack, and we started from the "correct" location. This means that if I had a RESOURCE filter on the stack, and I added a CONNECTION filter, the CONNECTION filter would be ignored. This should make sense, because we would insert the connection filter at the top of the c->output_filters list, but the end of r->output_filters pointed to the filter that used to be at the front of c->output_filters. This is obviously wrong. The new insertion code uses a doubly linked list. This has the advantage that we never lose a filter that has been inserted. Unfortunately, it comes with a separate set of headaches.

The problem is that we have two different cases were we use subrequests. The first is to insert more data into a response. The second is to replace the existing response with an internal redirect. These are two different cases and need to be treated as such.

In the first case, we are creating the subrequest from within a handler or filter. This means that the next filter should be passed to make_sub_request function, and the last resource filter in the sub-request will point to the next filter in the main request. This makes sense, because the sub-request's data needs to flow through the same set of filters as the main request. A graphical representation might help:

```
Default_handler --> includes_filter --> byterange --> content_length ->
etc
```

If the includes filter creates a sub request, then we don't want the data from that sub-request to go through the includes filter, because it might not be SSI data. So, the subrequest adds the following:

```
Default_handler --> includes_filter -/-> byterange --> content_length -> etc
                                    /
Default_handler --> sub_request_core
```

What happens if the subrequest is SSI data? Well, that's easy, the includes_filter is a resource filter, so it will be added to the sub request in between the Default_handler and the sub_request_core filter.

The second case for sub-requests is when one sub-request is going to become the real request. This happens whenever a sub-request is created outside of a handler or filter, and NULL is passed as the next filter to the make_sub_request function.

In this case, the resource filters no longer make sense for the new request, because the resource has changed. So, instead of starting from scratch, we simply point the front of the resource filters for the sub-request to the front of the protocol filters for the old request. This means that we won't lose any of the protocol filters, neither will we try to send this data through a filter that shouldn't see it.

The problem is that we are using a doubly-linked list for our filter stacks now. But, you should notice that it is possible for two lists to intersect in this model. So, you do you handle the previous pointer? This is a very difficult question to answer, because there is no "right" answer, either method is equally valid. I looked at why we use the previous pointer. The only reason for it is to allow for easier addition of new servers. With that being said, the solution I chose was to make the previous pointer always stay on the original request.

This causes some more complex logic, but it works for all cases. My concern in having it move to the sub-request, is that for the more common case (where a sub-request is used to add data to a response), the

main filter chain would be wrong. That didn't seem like a good idea to
me.

asis:
The final topic. :-) Mod_Asis is a bit of a hack, but the
handler needs to remove all filters except for connection filters, and
send the data. If you are using mod_asis, all other bets are off.

The absolutely last point is that the reason this code was so hard to
get right, was because we had hacked so much to force it to work. I
wrote most of the hacks originally, so I am very much to blame.
However, now that the code is right, I have started to remove some
hacks. Most people should have seen that the reset_filters and
add_required_filters functions are gone. Those inserted protocol level
filters for error conditions, in fact, both functions did the same
thing, one after the other, it was really strange. Because we don't
lose protocol filters for error cases any more, those hacks went away.
The HTTP_HEADER, Content-length, and Byterange filters are all added in
the insert_filters phase, because if they were added earlier, we had
some interesting interactions. Now, those could all be moved to be
inserted with the HTTP_IN, CORE, and CORE_IN filters. That would make
the code easier to follow.

Apache HTTP Server Version 2.0