Overview of new features in Apache HTTP Server 2.4

This document describes some of the major changes between the 2.2 and 2.4 versions of the Apache HTTP Server. For new features since version 2.0, see the 2.2 new features document.

Core Enhancements

Run-time Loadable MPMs
Multiple MPMs can now be built as loadable modules at compile time. The MPM of choice can be configured at run time via `LoadModule` directive.

Event MPM
The Event MPM is no longer experimental but is now fully supported.

Asynchronous support
Better support for asynchronous read/write for supporting MPMs and platforms.

Per-module and per-directory LogLevel configuration
The `LogLevel` can now be configured per module and per directory. New levels `trace1` to `trace8` have been added above the `debug` log level.

Per-request configuration sections
`<If>`, `<ElseIf>`, and `<Else>` sections can be used to set the configuration based on per-request criteria.

General-purpose expression parser
A new expression parser allows to specify complex conditions using a common syntax in directives like `SetEnvIfExpr`, `RewriteCond`, `Header`, `<If>`, and others.

KeepAliveTimeout in milliseconds
It is now possible to specify `KeepAliveTimeout` in milliseconds.

NameVirtualHost directive
No longer needed and is now deprecated.

Override Configuration
The new `AllowOverrideList` directive allows more fine grained control which directives are allowed in `.htaccess` files.

Config file variables
It is now possible to `Define` variables in the configuration, allowing a clearer representation if the same value is used at many places in the configuration.

Reduced memory usage
Despite many new features, 2.4.x tends to use less memory than 2.2.x.

New Modules

mod_proxy_fcgi
FastCGI Protocol backend for `mod_proxy`

mod_proxy_scgi
SCGI Protocol backend for `mod_proxy`

mod_proxy_express
Provides dynamically configured mass reverse proxies for `mod_proxy`

mod_remoteip
Replaces the apparent client remote IP address and hostname for the request with the IP address list presented by a proxies or a load balancer

via the request headers.

mod_heartmonitor, **mod_lbmethod_heartbeat**
: Allow mod_proxy_balancer to base loadbalancing decisions on the number of active connections on the backend servers.

mod_proxy_html
: Formerly a third-party module, this supports fixing of HTML links in a reverse proxy situation, where the backend generates URLs that are not valid for the proxy's clients.

mod_sed
: An advanced replacement of mod_substitute, allows to edit the response body with the full power of sed.

mod_auth_form
: Enables form-based authentication.

mod_session
: Enables the use of session state for clients, using cookie or database storage.

mod_allowmethods
: New module to restrict certain HTTP methods without interfering with authentication or authorization.

mod_lua
: Embeds the Lua language into httpd, for configuration and small business logic functions. (Experimental)

mod_log_debug
: Allows the addition of customizable debug logging at different phases of the request processing.

mod_buffer
: Provides for buffering the input and output filter stacks

mod_data
: Convert response body into an RFC2397 data URL

mod_ratelimit
: Provides Bandwidth Rate Limiting for Clients

mod_request
: Provides Filters to handle and make available HTTP request bodies

mod_reflector
: Provides Reflection of a request body as a response via the output filter stack.

mod_slotmem_shm
: Provides a Slot-based shared memory provider (ala the scoreboard).

mod_xml2enc
: Formerly a third-party module, this supports internationalisation in libxml2-based (markup-aware) filter modules.

mod_macro **(available since 2.4.5)**
: Provide macros within configuration files.

mod_proxy_wstunnel **(available since 2.4.5)**
: Support web-socket tunnels.

mod_authnz_fcgi **(available since 2.4.10)**
: Enable FastCGI authorizer applications to authenticate and/or authorize clients.

mod_http2 **(available since 2.4.17)**
: Support for the HTTP/2 transport layer.

mod_proxy_http2 **(available since 2.4.19)**
: HTTP/2 Protocol backend for mod_proxy

mod_proxy_hcheck (available since 2.4.21)
Support independent dynamic health checks for remote proxiy backend servers.

mod_brotli (available since 2.4.26)
Support the Brotli compression algorithm.

mod_md (available since 2.4.30)
Support the ACME protocol to automate certificate provisionning.

mod_socache_redis (available since 2.4.39)
Support Redis based shared object cache provider.

mod_systemd (available since 2.4.42)
systemd integration. It allows httpd to be used in a service with the systemd `Type=notify`.

Module Enhancements

mod_ssl
`mod_ssl` can now be configured to use an OCSP server to check the validation status of a client certificate. The default responder is configurable, along with the decision on whether to prefer the responder designated in the client certificate itself.
`mod_ssl` now also supports OCSP stapling, where the server pro-actively obtains an OCSP verification of its certificate and transmits that to the client during the handshake.
`mod_ssl` can now be configured to share SSL Session data between servers through memcached
EC keys are now supported in addition to RSA and DSA.
Support for TLS-SRP (available in 2.4.4 and later).

mod_proxy
The `ProxyPass` directive is now most optimally configured within a `Location` or `LocationMatch` block, and offers a significant performance advantage over the traditional two-parameter syntax when present in large numbers.
The source address used for proxy requests is now configurable.
Support for Unix domain sockets to the backend (available in 2.4.7 and later).

mod_proxy_balancer
More runtime configuration changes for BalancerMembers via balancer-manager
Additional BalancerMembers can be added at runtime via balancer-manager
Runtime configuration of a subset of Balancer parameters
BalancerMembers can be set to 'Drain' so that they only respond to existing sticky sessions, allowing them to be taken gracefully offline.
Balancer settings can be persistent after restarts.

mod_cache
The `mod_cache` CACHE filter can be optionally inserted at a given point in the filter chain to provide fine control over caching.
`mod_cache` can now cache HEAD requests.
Where possible, `mod_cache` directives can now be set per directory, instead of per server.
The base URL of cached URLs can be customised, so that a cluster of caches can share the same endpoint URL prefix.
`mod_cache` is now capable of serving stale cached data when a backend is unavailable (error 5xx).
`mod_cache` can now insert HIT/MISS/REVALIDATE into an X-Cache header.

mod_include

Support for the 'onerror' attribute within an 'include' element, allowing an error document to be served on error instead of the default error string.

mod_cgi, mod_include, mod_isapi, ...
Translation of headers to environment variables is more strict than before to mitigate some possible cross-site-scripting attacks via header injection. Header names containing invalid characters (including underscores) are no longer converted to environment variables. Environment Variables in Apache has some pointers on how to work around broken legacy clients which require such headers. (This affects all modules which use these environment variables.)

mod_authz_core Authorization Logic Containers
Advanced authorization logic may now be specified using the `Require` directive and the related container directives, such as `<RequireAll>`.

mod_rewrite
`mod_rewrite` adds the `[QSD]` (Query String Discard) and `[END]` flags for `RewriteRule` to simplify common rewriting scenarios.
Adds the possibility to use complex boolean expressions in `RewriteCond`.
Allows the use of SQL queries as `RewriteMap` functions.

mod_ldap, mod_authnz_ldap
`mod_authnz_ldap` adds support for nested groups.
`mod_ldap` adds `LDAPConnectionPoolTTL`, `LDAPTimeout`, and other improvements in the handling of timeouts. This is especially useful for setups where a stateful firewall drops idle connections to the LDAP server.
`mod_ldap` adds `LDAPLibraryDebug` to log debug information provided by the used LDAP toolkit.

mod_info
`mod_info` can now dump the pre-parsed configuration to stdout during server startup.

mod_auth_basic
New generic mechanism to fake basic authentication (available in 2.4.5 and later).

Program Enhancements

fcgistarter
New FastCGI daemon starter utility

htcacheclean
Current cached URLs can now be listed, with optional metadata included.
Allow explicit deletion of individual cached URLs from the cache.
File sizes can now be rounded up to the given block size, making the size limits map more closely to the real size on disk.
Cache size can now be limited by the number of inodes, instead of or in addition to being limited by the size of the files on disk.

rotatelogs
May now create a link to the current log file.
May now invoke a custom post-rotate script.

htpasswd, htdbm
Support for the bcrypt algorithm (available in 2.4.4 and later).

Documentation

mod_rewrite
The `mod_rewrite` documentation has been rearranged and almost completely rewritten, with a focus on examples and common usage, as well as on showing you when other solutions are more appropriate. The Rewrite Guide is now a top-level section with much more detail and better organization.

mod_ssl

The mod_ssl documentation has been greatly enhanced, with more examples at the getting started level, in addition to the previous focus on technical details.

Caching Guide

The Caching Guide has been rewritten to properly distinguish between the RFC2616 HTTP/1.1 caching features provided by mod_cache, and the generic key/value caching provided by the socache interface, as well as to cover specialised caching provided by mechanisms such as mod_file_cache.

Module Developer Changes

Check Configuration Hook Added

A new hook, check_config, has been added which runs between the pre_config and open_logs hooks. It also runs before the test_config hook when the -t option is passed to httpd. The check_config hook allows modules to review interdependent configuration directive values and adjust them while messages can still be logged to the console. The user can thus be alerted to misconfiguration problems before the core open_logs hook function redirects console output to the error log.

Expression Parser Added

We now have a general-purpose expression parser, whose API is exposed in *ap_expr.h*. This is adapted from the expression parser previously implemented in mod_ssl.

Authorization Logic Containers

Authorization modules now register as a provider, via ap_register_auth_provider(), to support advanced authorization logic, such as <RequireAll>.

Small-Object Caching Interface

The *ap_socache.h* header exposes a provider-based interface for caching small data objects, based on the previous implementation of the mod_ssl session cache. Providers using a shared-memory cyclic buffer, disk-based dbm files, and a memcache distributed cache are currently supported.

Cache Status Hook Added

The mod_cache module now includes a new cache_status hook, which is called when the caching decision becomes known. A default implementation is provided which adds an optional X-Cache and X-Cache-Detail header to the response.

The developer documentation contains a detailed list of API changes.

Overview of new features in Apache HTTP Server 2.2

This document describes some of the major changes between the 2.0 and 2.2 versions of the Apache HTTP Server. For new features since version 1.3, see the 2.0 new features document.

Core Enhancements

Authn/Authz

The bundled authentication and authorization modules have been refactored. The new mod_authn_alias(already removed from 2.3/2.4) module can greatly simplify certain authentication configurations. See module name changes, and the developer changes for more information about how these changes affects users and module writers.

Caching

mod_cache, mod_cache_disk, and mod_mem_cache(already removed from 2.3/2.4) have undergone a lot of changes, and are now considered production-quality. htcacheclean has been introduced to clean up mod_cache_disk setups.

Configuration

The default configuration layout has been simplified and modularised. Configuration snippets which can be used to enable commonly-used features are now bundled with Apache, and can be easily added to the main server config.

Graceful stop

The prefork, worker and event MPMs now allow httpd to be shutdown gracefully via the graceful-stop signal. The GracefulShutdownTimeout directive has been added to specify an optional timeout, after which httpd will terminate regardless of the status of any requests being served.

Proxying

The new mod_proxy_balancer module provides load balancing services for mod_proxy. The new mod_proxy_ajp module adds support for the `Apache JServ Protocol version 1.3` used by Apache Tomcat.

Regular Expression Library Updated

Version 5.0 of the Perl Compatible Regular Expression Library (PCRE) is now included. httpd can be configured to use a system installation of PCRE by passing the `--with-pcre` flag to configure.

Smart Filtering

mod_filter introduces dynamic configuration to the output filter chain. It enables filters to be conditionally inserted, based on any Request or Response header or environment variable, and dispenses with the more problematic dependencies and ordering problems in the 2.0 architecture.

Large File Support

httpd is now built with support for files larger than 2GB on modern 32-bit Unix systems. Support for handling >2GB request bodies has also been added.

Event MPM

The event MPM uses a separate thread to handle Keep Alive requests and accepting connections. Keep Alive requests have traditionally required httpd to dedicate a worker to handle it. This dedicated worker could not be used again until the Keep Alive timeout was reached.

SQL Database Support

mod_dbd, together with the apr_dbd framework, brings direct SQL support to modules that need it. Supports connection pooling in threaded MPMs.

Module Enhancements

Authn/Authz

Modules in the aaa directory have been renamed and offer better support for digest authentication. For example, mod_auth is now split into mod_auth_basic and mod_authn_file; mod_auth_dbm is now called mod_authn_dbm; mod_access has been renamed mod_authz_host. There is also a new mod_authn_alias(already removed from 2.3/2.4) module for simplifying certain authentication configurations.

mod_authnz_ldap

This module is a port of the 2.0 mod_auth_ldap module to the 2.2 Authn/Authz framework. New features include using LDAP attribute values and complicated search filters in the Require directive.

mod_authz_owner

A new module that authorizes access to files based on the owner of the file on the file system

mod_version

A new module that allows configuration blocks to be enabled based on the version number of the running server.

mod_info

Added a new ?config argument which will show the configuration directives as parsed by Apache, including their file name and line number. The module also shows the order of all request hooks and additional build information, similar to httpd -V.

mod_ssl

Added a support for RFC 2817, which allows connections to upgrade from clear text to TLS encryption.

mod_imagemap

mod_imap has been renamed to mod_imagemap to avoid user confusion.

Program Enhancements

httpd

A new command line option -M has been added that lists all modules that are loaded based on the current configuration. Unlike the -l option, this list includes DSOs loaded via mod_so.

httxt2dbm

A new program used to generate dbm files from text input, for use in RewriteMap with the dbm map type.

Module Developer Changes

APR 1.0 API

Apache 2.2 uses the APR 1.0 API. All deprecated functions and symbols have been removed from APR and APR-Util. For details, see the APR Website.

Authn/Authz

The bundled authentication and authorization modules have been renamed along the following lines:

- mod_auth_* -> Modules that implement an HTTP authentication mechanism

- `mod_authn_*` -> Modules that provide a backend authentication provider
- `mod_authz_*` -> Modules that implement authorization (or access)
- `mod_authnz_*` -> Module that implements both authentication & authorization

There is a new authentication backend provider scheme which greatly eases the construction of new authentication backends.

Connection Error Logging

A new function, `ap_log_cerror` has been added to log errors that occur with the client's connection. When logged, the message includes the client IP address.

Test Configuration Hook Added

A new hook, `test_config` has been added to aid modules that want to execute special code only when the user passes `-t` to `httpd`.

Set Threaded MPM's Stacksize

A new directive, `ThreadStackSize` has been added to set the stack size on all threaded MPMs. This is required for some third-party modules on platforms with small default thread stack size.

Protocol handling for output filters

In the past, every filter has been responsible for ensuring that it generates the correct response headers where it affects them. Filters can now delegate common protocol management to `mod_filter`, using the `ap_register_output_filter_protocol` or `ap_filter_protocol` calls.

Monitor hook added

Monitor hook enables modules to run regular/scheduled jobs in the parent (root) process.

Regular expression API changes

The `pcreposix.h` header is no longer available; it is replaced by the new `ap_regex.h` header. The POSIX.2 `regex.h` implementation exposed by the old header is now available under the `ap_` namespace from `ap_regex.h`. Calls to `regcomp`, `regexec` and so on can be replaced by calls to `ap_regcomp`, `ap_regexec`.

DBD Framework (SQL Database API)

With Apache 1.x and 2.0, modules requiring an SQL backend had to take responsibility for managing it themselves. Apart from reinventing the wheel, this can be very inefficient, for example when several modules each maintain their own connections.

Apache 2.1 and later provides the `ap_dbd` API for managing database connections (including optimised strategies for threaded and unthreaded MPMs), while APR 1.2 and later provides the `apr_dbd` API for interacting with the database.

New modules SHOULD now use these APIs for all SQL database operations. Existing applications SHOULD be upgraded to use it where feasible, either transparently or as a recommended option to their users.

Overview of new features in Apache HTTP Server 2.0

This document describes some of the major changes between the 1.3 and 2.0 versions of the Apache HTTP Server.

Core Enhancements

Unix Threading
On Unix systems with POSIX threads support, Apache httpd 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 httpd's configuration system more similar to that of other packages.

Multiprotocol Support
Apache HTTP Server 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 HTTP Server 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 httpd 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 HTTP Server.

IPv6 Support
On systems where IPv6 is supported by the underlying Apache Portable Runtime library, Apache httpd gets IPv6 listening sockets by default. Additionally, the `Listen`, `NameVirtualHost`, and `VirtualHost` directives support IPv6 numeric address strings (e.g., "`Listen [2001:db8::1]:8080`").

Filtering
Apache httpd 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`. The module `mod_ext_filter` allows external programs to act as filters in much the same way that CGI programs can act as handlers.

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 httpd 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.*

Regular Expression Library Updated
Apache httpd 2.0 includes the Perl Compatible Regular Expression Library (PCRE). All regular expression evaluation now uses the more powerful Perl 5 syntax.

Module Enhancements

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

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

mod_deflate
New module in Apache httpd 2.0. This module allows supporting browsers to request that content be compressed before delivery, saving network bandwidth.

mod_auth_ldap
New module in Apache httpd 2.0.41. This module allows an LDAP database to be used to store credentials for HTTP Basic Authentication. A companion module, mod_ldap provides connection pooling and results caching.

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

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

mod_file_cache
New module in Apache httpd 2.0. This module includes the functionality of mod_mmap_static in Apache HTTP Server version 1.3, plus adds further caching abilities.

mod_headers
This module is much more flexible in Apache httpd 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.

Upgrading to 2.4 from 2.2

In order to assist folks upgrading, we maintain a document describing information critical to existing Apache HTTP Server 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. Application and module developers can find a summary of API changes in the API updates overview.

This document describes changes in server behavior that might require you to change your configuration or how you use the server in order to continue using 2.4 as you are currently using 2.2. To take advantage of new features in 2.4, see the New Features document.

This document describes only the changes from 2.2 to 2.4. If you are upgrading from version 2.0, you should also consult the 2.0 to 2.2 upgrading document.

Compile-Time Configuration Changes

The compilation process is very similar to the one used in version 2.2. Your old `configure` command line (as found in `build/config.nice` in the installed server directory) can be used in most cases. There are some changes in the default settings. Some details of changes:

- These modules have been removed: mod_authn_default, mod_authz_default, mod_mem_cache. If you were using mod_mem_cache in 2.2, look at `mod_cache_disk` in 2.4.
- All load balancing implementations have been moved to individual, self-contained mod_proxy submodules, e.g. `mod_lbmethod_bybusyness`. You might need to build and load any of these that your configuration uses.
- Platform support has been removed for BeOS, TPF, and even older platforms such as A/UX, Next, and Tandem. These were believed to be broken anyway.
- configure: dynamic modules (DSO) are built by default
- configure: By default, only a basic set of modules is loaded. The other `LoadModule` directives are commented out in the configuration file.
- configure: the "most" module set gets built by default
- configure: the "reallyall" module set adds developer modules to the "all" set

Run-Time Configuration Changes

There have been significant changes in authorization configuration, and other minor configuration changes, that could require changes to your 2.2 configuration files before using them for 2.4.

Authorization

Any configuration file that uses authorization will likely need changes.

You should review the Authentication, Authorization and Access Control Howto, especially the section Beyond just authorization which explains the new mechanisms for controlling the order in which the authorization directives are applied.

Directives that control how authorization modules respond when they don't match the authenticated user have been removed: This includes AuthzLDAPAuthoritative, AuthzDBDAuthoritative, AuthzDBMAuthoritative, AuthzGroupFileAuthoritative, AuthzUserAuthoritative, and AuthzOwnerAuthoritative. These directives have been replaced by the more expressive `RequireAny`, `RequireNone`, and `RequireAll`.

If you use mod_authz_dbm, you must port your configuration to use `Require dbm-group ...` in place of `Require group ...`.

Access control

In 2.2, access control based on client hostname, IP address, and other characteristics of client requests was done using the directives Order, Allow, Deny, and Satisfy.

In 2.4, such access control is done in the same way as other authorization checks, using the new module mod_authz_host. The old access control idioms should be replaced by the new authentication mechanisms, although for compatibility with old configurations, the new module mod_access_compat is provided.

> **Mixing old and new directives**
>
> Mixing old directives like Order, Allow or Deny with new ones like Require is technically possible but discouraged. mod_access_compat was created to support configurations containing only old directives to facilitate the 2.4 upgrade. Please check the examples below to get a better idea about issues that might arise.

Here are some examples of old and new ways to do the same access control.

In this example, there is no authentication and all requests are denied.

2.2 configuration:

```
Order deny,allow
Deny from all
```

2.4 configuration:

```
Require all denied
```

In this example, there is no authentication and all requests are allowed.

2.2 configuration:

```
Order allow,deny
Allow from all
```

2.4 configuration:

```
Require all granted
```

In the following example, there is no authentication and all hosts in the example.org domain are allowed access; all other hosts are denied access.

2.2 configuration:

```
Order Deny,Allow
Deny from all
Allow from example.org
```

2.4 configuration:

```
Require host example.org
```

In the following example, mixing old and new directives leads to unexpected results.

Mixing old and new directives: NOT WORKING AS EXPECTED

```
DocumentRoot "/var/www/html"

<Directory "/">
```

```
    AllowOverride None
    Order deny,allow
    Deny from all
</Directory>

<Location "/server-status">
    SetHandler server-status
    Require local
</Location>

access.log - GET /server-status 403 127.0.0.1
error.log - AH01797: client denied by server configuration
```

Why httpd denies access to servers-status even if the configuration seems to allow it? Because `mod_access_compat` directives take precedence over the `mod_authz_host` one in this configuration merge scenario.

This example conversely works as expected:

Mixing old and new directives: WORKING AS EXPECTED

```
DocumentRoot "/var/www/html"

<Directory "/">
    AllowOverride None
    Require all denied
</Directory>

<Location "/server-status">
    SetHandler server-status
    Order deny,allow
    Deny from all
    Allow From 127.0.0.1
</Location>

access.log - GET /server-status 200 127.0.0.1
```

So even if mixing configuration is still possible, please try to avoid it when upgrading: either keep old directives and then migrate to the new ones on a later stage or just migrate everything in bulk.

In many configurations with authentication, where the value of the `Satisfy` was the default of *ALL*, snippets that simply disabled host-based access control are omitted:

2.2 configuration:

```
# 2.2 config that disables host-based access control and u
Order Deny,Allow
Allow from all
AuthType Basic
AuthBasicProvider file
AuthUserFile /example.com/conf/users.passwd
AuthName secure
Require valid-user
```

2.4 configuration:

```
# No replacement of disabling host-based access control ne
AuthType Basic
AuthBasicProvider file
AuthUserFile /example.com/conf/users.passwd
AuthName secure
Require valid-user
```

In configurations where both authentication and access control were meaningfully combined, the access control directives should be migrated. This example allows requests meeting *both* criteria:

2.2 configuration:

```
Order allow,deny
Deny from all
# Satisfy ALL is the default
Satisfy ALL
Allow from 127.0.0.1
AuthType Basic
AuthBasicProvider file
AuthUserFile /example.com/conf/users.passwd
AuthName secure
Require valid-user
```

2.4 configuration:

```
AuthType Basic
AuthBasicProvider file
AuthUserFile /example.com/conf/users.passwd
AuthName secure
<RequireAll>
  Require valid-user
  Require ip 127.0.0.1
</RequireAll>
```

In configurations where both authentication and access control were meaningfully combined, the access control directives should be migrated. This example allows requests meeting *either* criteria:

2.2 configuration:

```
Order allow,deny
Deny from all
Satisfy any
Allow from 127.0.0.1
AuthType Basic
AuthBasicProvider file
AuthUserFile /example.com/conf/users.passwd
AuthName secure
Require valid-user
```

2.4 configuration:

```
AuthType Basic
AuthBasicProvider file
AuthUserFile /example.com/conf/users.passwd
AuthName secure
# Implicitly <RequireAny>
Require valid-user
Require ip 127.0.0.1
```

Other configuration changes

Some other small adjustments may be necessary for particular configurations as discussed below.

- `MaxRequestsPerChild` has been renamed to `MaxConnectionsPerChild`, describes more accurately what it does. The old name is still supported.
- `MaxClients` has been renamed to `MaxRequestWorkers`, which describes more accurately what it does. For async MPMs, like `event`, the maximum number of clients is not equivalent than the number of worker threads. The old name is still supported.
- The `DefaultType` directive no longer has any effect, other than to emit a warning if it's used with any value other than `none`. You need to use other configuration settings to replace it in 2.4.
- `AllowOverride` now defaults to `None`.
- `EnableSendfile` now defaults to Off.
- `FileETag` now defaults to "MTime Size" (without INode).
- `mod_dav_fs`: The format of the `DavLockDB` file has changed for systems with inodes. The old `DavLockDB` file must be deleted on upgrade.

- KeepAlive only accepts values of On or Off. Previously, any value other than "Off" or "0" was treated as "On".
- Directives AcceptMutex, LockFile, RewriteLock, SSLMutex, SSLStaplingMutex, and WatchdogMutexPath have been replaced with a single Mutex directive. You will need to evaluate any use of these removed directives in your 2.2 configuration to determine if they can just be deleted or will need to be replaced using Mutex.
- mod_cache: CacheIgnoreURLSessionIdentifiers now does an exact match against the query string instead of a partial match. If your configuration was using partial strings, e.g. using sessionid to match /someapplication/image.gif;jsessionid=123456789, then you will need to change to the full string jsessionid.
- mod_cache: The second parameter to CacheEnable only matches forward proxy content if it begins with the correct protocol. In 2.2 and earlier, a parameter of '/' matched all content.
- mod_ldap: LDAPTrustedClientCert is now consistently a per-directory setting only. If you use this directive, review your configuration to make sure it is present in all the necessary directory contexts.
- mod_filter: FilterProvider syntax has changed and now uses a boolean expression to determine if a filter is applied.
- mod_include:
  - The #if expr element now uses the new expression parser. The old syntax can be restored with the new directive SSILegacyExprParser.
  - An SSI* config directive in directory scope no longer causes all other per-directory SSI* directives to be reset to their default values.
- mod_charset_lite: The DebugLevel option has been removed in favour of per-module LogLevel configuration.
- mod_ext_filter: The DebugLevel option has been removed in favour of per-module LogLevel configuration.
- mod_proxy_scgi: The default setting for PATH_INFO has changed from httpd 2.2, and some web applications will no longer operate properly with the new PATH_INFO setting. The previous setting can be restored by configuring the proxy-scgi-pathinfo variable.
- mod_ssl: CRL based revocation checking now needs to be explicitly configured through SSLCARevocationCheck.
- mod_substitute: The maximum line length is now limited to 1MB.
- mod_reqtimeout: If the module is loaded, it will now set some default timeouts.
- mod_dumpio: DumpIOLogLevel is no longer supported. Data is always logged at LogLevel trace7.
- On Unix platforms, piped logging commands configured using either ErrorLog or CustomLog were invoked using /bin/sh -c in 2.2 and earlier. In 2.4 and later, piped logging commands are executed directly. To restore the old behaviour, see the piped logging documentation.

Misc Changes

- mod_autoindex: will now extract titles and display descriptions for .xhtml files, which were previously ignored.
- mod_ssl: The default format of the *_DN variables has changed. The old format can still be used with the new LegacyDNStringFormat argument to SSLOptions. The SSLv2 protocol is no longer supported. SSLProxyCheckPeerCN and SSLProxyCheckPeerExpire now default to On, causing proxy requests to HTTPS hosts with bad or outdated certificates to fail with a 502 status code (Bad gateway)
- htpasswd now uses MD5 hash by default on all platforms.
- The NameVirtualHost directive no longer has any effect, other than to emit a warning. Any address/port combination appearing in multiple virtual

hosts is implicitly treated as a name-based virtual host.
- mod_deflate will now skip compression if it knows that the size overhead added by the compression is larger than the data to be compressed.
- Multi-language error documents from 2.2.x may not work unless they are adjusted to the new syntax of mod_include's `#if expr=` element or the directive SSILegacyExprParser is enabled for the directory containing the error documents.
- The functionality provided by `mod_authn_alias` in previous versions (i.e., the AuthnProviderAlias directive) has been moved into mod_authn_core.
- mod_cgid uses the servers Timeout to limit the length of time to wait for CGI output. This timeout can be overridden with CGIDScriptTImeout.

Third Party Modules

All modules must be recompiled for 2.4 before being loaded.

Many third-party modules designed for version 2.2 will otherwise work unchanged with the Apache HTTP Server version 2.4. Some will require changes; see the API update overview.

Common problems when upgrading

- Startup errors:
  - `Invalid command 'User', perhaps misspelled or defined by a module not included in the server configuration` - load module mod_unixd
  - `Invalid command 'Require', perhaps misspelled or defined by a module not included in the server configuration`, or `Invalid command 'Order', perhaps misspelled or defined by a module not included in the server configuration` - load module mod_access_compat, or update configuration to 2.4 authorization directives.
  - `Ignoring deprecated use of DefaultType in line NN of /path/to/httpd.conf` - remove DefaultType and replace with other configuration settings.
  - `Invalid command 'AddOutputFilterByType', perhaps misspelled or defined by a module not included in the server configuration` - AddOutputFilterByType has moved from the core to mod_filter, which must be loaded.
- Errors serving requests:
  - `configuration error: couldn't check user: /path` - load module mod_authn_core.
  - `.htaccess` files aren't being processed - Check for an appropriate AllowOverride directive; the default changed to `None` in 2.4.

The Apache License, Version 2.0

Available Languages: **en**

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. **Definitions**

   "License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

   "Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

   "Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

   "Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

   "Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

   "Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

   "Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. **Grant of Copyright License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.
3. **Grant of Patent License.** Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent

infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. **Redistribution.** You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:
    a. You must give any other recipients of the Work or Derivative Works a copy of this License; and
    b. You must cause any modified files to carry prominent notices stating that You changed the files; and
    c. You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and
    d. If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

    You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. **Submission of Contributions.** Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.
6. **Trademarks.** This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.
7. **Disclaimer of Warranty.** Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.
8. **Limitation of Liability.** In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.
9. **Accepting Warranty or Additional Liability.** While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS

APPENDIX: How to apply the Apache License to your work.

To apply the Apache License to your work, attach the following boilerplate notice, with the fields enclosed by brackets "[]" replaced with your own identifying information. (Don't include the brackets!) The text should be enclosed in the appropriate comment syntax for the file format. We also recommend that a file or class name and description of purpose be included on the same "printed page" as the copyright notice for easier identification within third-party archives.

Compiling and Installing

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

Apache httpd uses `libtool` and `autoconf` to create a build environment that looks like many other Open Source projects.

If you are upgrading from one minor version to the next (for example, 2.4.8 to 2.4.9), please skip down to the upgrading section.

Overview for the impatient

Installing on Fedora/CentOS/Red Hat Enterprise Linux

```
sudo dnf install httpd
sudo service httpd start
```

Older releases of these distros use `yum` rather than `dnf`. See the Fedora project's documentation for platform-specific notes.

Installing on Ubuntu/Debian

```
sudo apt install apache2
sudo service apache2 start
```

See Ubuntu's documentation for platform-specific notes.

Installing from source

Download	Download the latest release from http://httpd.apache.org/download.cgi

Extract	`$ gzip -d httpd-NN.tar.gz`
	`$ tar xvf httpd-NN.tar`
	`$ cd httpd-NN`

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

NN must be replaced with the current 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.

Don't see your favorite platform mentioned here? Come help us improve this doc.

Requirements

The following requirements exist for building Apache httpd:

APR and APR-Util

Make sure you have APR and APR-Util already installed on your system. If you don't, or prefer to not use the system-provided versions, download the latest versions of both APR and APR-Util from Apache APR, unpack them into `/httpd_source_tree_root/srclib/apr` and `/httpd_source_tree_root/srclib/apr-util` (be sure the directory names do not have version numbers; for example, the APR distribution must be under /httpd_source_tree_root/srclib/apr/) and use `./configure`'s `--with-included-apr` option. On some platforms, you may have to install the corresponding `-dev` packages to allow httpd to build against your installed copy of APR and APR-Util.

Perl-Compatible Regular Expressions Library (PCRE)

This library is required but not longer bundled with httpd. Download the source code from http://www.pcre.org, or install a Port or Package. If your build system can't find the pcre-config script installed by the PCRE build, point to it using the `--with-pcre` parameter. On some platforms, you may have to install the corresponding `-dev` package to allow httpd to build against your installed copy of PCRE.

Disk Space

Make sure you have at least 50 MB of temporary free disk space available. After installation the server occupies approximately 10 MB of disk space. The actual disk space requirements will vary considerably based on your chosen configuration options, any third-party modules, and, of course, the size of the web site or sites that you have on the server.

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. 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 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 or newer are sufficient). If no Perl 5 interpreter is found by the `configure` script, you will not be able to use the affected support scripts. Of course, you will still be able to build and use Apache httpd.

Download

The Apache HTTP Server can be downloaded from the Apache HTTP Server download site, which lists several mirrors. Most users of Apache on unix-like systems will be better off downloading and compiling a source version. The build process (described below) is easy, and it allows you to customize your server to suit your needs. In addition, binary releases are often not up to date with the latest source releases. If you do download a binary, follow the instructions in the `INSTALL.bindist` file inside the distribution.

After downloading, 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. Details on how to do this are available on the download page and an extended example is available describing the use of PGP.

Extract

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

```
$ gzip -d httpd-NN.tar.gz
$ tar xvf httpd-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 an unreleased 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.

The most important option is the location `--prefix` where Apache is to be installed later, because Apache has to be configured for this location to work correctly. More fine-tuned control of the location of files is possible with additional configure options.

Also at this point, you can specify which features you want included in Apache by enabling and disabling modules. Apache comes with a wide range of modules included by default. They will be compiled as shared objects (DSOs) which can be loaded or unloaded at runtime. You can also choose to compile modules statically by using the option `--enable-module=static`.

Additional modules are enabled using the `--enable-module` option, where *module* is the name of the module with the `mod_` string removed and with any underscore converted to a dash. Similarly, you can disable modules with the `--disable-module` option. Be careful when using these options, since `configure` cannot warn you if the module you specify does not exist; it will simply ignore the option.

In addition, it is sometimes necessary to provide the `configure` script with extra information about the location of your compiler, libraries, or header files. This is done by passing either environment variables or command line options to `configure`. For more information, see the `configure` manual page. Or invoke `configure` using the `--help` option.

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_ldap` and `mod_lua`:

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

Details on all the different `configure` options are available on the `configure` manual page.

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 several minutes to compile and the time will vary widely depending on your hardware and the number of modules that you have enabled.

Install

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

```
$ make install
```

This step will typically require root privileges, since *PREFIX* is usually a directory with restricted write permissions.

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 *PREFIX*`/docs/manual/` or consult http://httpd.apache.org/docs/trunk/ for the most recent version of this manual and a complete reference of available configuration directives.

Test

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

```
$ PREFIX/bin/apachectl -k start
```

You should then be able to request your first document via the 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 -k stop
```

Upgrading

The first step in upgrading is to read the release announcement and the file `CHANGES` in the source distribution to find any changes that may affect your site. When changing between major releases (for example, from 2.0 to 2.2 or from 2.2 to 2.4), there will likely be major differences in the compile-time and run-time configuration that will require manual adjustments. All modules will also need to be upgraded to accommodate changes in the module API.

Upgrading from one minor version to the next (for example, from 2.2.55 to 2.2.57) is easier. The `make install` process will not overwrite any of your existing documents, log files, or configuration files. In addition, the developers make every effort to avoid incompatible changes in the `configure` options, run-time configuration, or the module API between minor versions. In most cases you should be able to use an identical `configure` command line, an identical configuration file, and all of your modules should continue to work.

To upgrade across minor versions, start by finding the file `config.nice` in the `build` directory of your installed server or at the root of the source tree for your old install. This will contain the exact `configure` command line that you used to configure the source tree. Then to upgrade from one version to the next, you need only copy the `config.nice` file to the source tree of the new version, edit it to make any desired changes, and then run:

```
$ ./config.nice
$ make
$ make install
$ PREFIX/bin/apachectl -k graceful-stop
$ PREFIX/bin/apachectl -k start
```

You should always test any new version in your environment before putting it into production. For example, you can install and run the new version along side the old one by using a different `--prefix` and a different port (by adjusting the `Listen` directive) to test for any incompatibilities before doing the final upgrade.

You can pass additional arguments to `config.nice`, which will be appended to your original `configure` options:

```
$ ./config.nice --prefix=/home/test/apache --with-port=90
```

Third-party packages

A large number of third parties provide their own packaged distributions of the Apache HTTP Server for installation on particular platforms. This includes the various Linux distributions, various third-party Windows packages, Mac OS X, Solaris, and many more.

Our software license not only permits, but encourages, this kind of redistribution. However, it does result in a situation where the configuration layout and defaults on your installation of the server may differ from what is stated in the documentation. While unfortunate, this situation is not likely to change any time soon.

A description of these third-party distributions is maintained in the HTTP Server wiki, and should reflect the current state of these third-party distributions. However, you will need to familiarize yourself with your particular platform's package management and installation procedures.

Starting Apache

On Windows, Apache is normally run as a service. For details, see Running Apache as a Service.

On Unix, the httpd program is run as a daemon that executes continuously in the background to handle requests. This document describes how to invoke httpd.

How Apache Starts

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 recommended method of invoking the httpd executable is to use the apachectl control script. This script sets certain environment variables that are necessary for httpd to function correctly under some operating systems, and then invokes the httpd binary. apachectl will pass through any command line arguments, so any httpd options may also be used with apachectl. You may also directly edit the apachectl script by changing the HTTPD variable near the top to specify the correct location of the httpd binary and any command-line arguments that you wish to be *always* present.

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/apache2/bin/apachectl -f
/usr/local/apache2/conf/httpd.conf
```

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.

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 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 to act like a standard SysV init script; it can take the arguments `start`, `restart`, and `stop` and translate them into the appropriate signals to `httpd`. So you can often simply link `apachectl` into the appropriate init directory. 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.

Stopping and Restarting Apache HTTP Server

This document covers stopping and restarting Apache HTTP Server on Unix-like systems. Windows NT, 2000 and XP users should see Running httpd as a Service and Windows 9x and ME users should see Running httpd as a Console Application for information on how to control httpd on those platforms.

Introduction

In order to stop or restart the Apache HTTP Server, you must send a signal to the running `httpd` processes. There are two ways to send the signals. First, you can use the unix `kill` command to directly send signals to the processes. 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 four signals that you can send the parent: `TERM`, `USR1`, `HUP`, and `WINCH`, 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/apache2/logs/httpd.pid`
```

The second method of signaling the `httpd` processes is to use the `-k` command line options: `stop`, `restart`, `graceful` and `graceful-stop`, as described below. These are arguments to the `httpd` binary, but we recommend that you send them using the `apachectl` control script, which will pass them through to `httpd`.

After you have signaled `httpd`, you can read about its progress by issuing:

```
tail -f /usr/local/apache2/logs/error_log
```

Modify those examples to match your `ServerRoot` and `PidFile` settings.

Stop Now

Signal: TERM

```
apachectl -k stop
```

Sending the `TERM` or `stop` 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 -k graceful
```

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

This code is designed to always respect the process control directive of the MPMs, so the number of processes and threads available to serve clients will be maintained at the appropriate values throughout the restart process. 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. Hence 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 mod_status 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.

> When you issue a restart, a syntax check is first run, to ensure that there are no errors in the configuration files. If your configuration file has errors in it, you will get an error message about that syntax error, and the server will refuse to restart. This avoids the situation where the server halts and then cannot restart, leaving you with a non-functioning server.
>
> 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 -k restart
```

Sending the HUP or restart 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 mod_status will notice that the server statistics are set to zero when a HUP is sent.

> As with a graceful restart, a syntax check is run before the restart is attempted. If your configuration file has errors in it, the restart will not be attempted, and you will receive notification of the syntax error(s).

Graceful Stop

Signal: WINCH

```
apachectl -k graceful-stop
```

The WINCH or graceful-stop 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 will then remove its PidFile and cease listening

on all ports. The parent will continue to run, and monitor children which are handling requests. Once all children have finalised and exited or the timeout specified by the `GracefulShutdownTimeout` has been reached, the parent will also exit. If the timeout is reached, any remaining children will be sent the `TERM` signal to force them to exit.

A `TERM` signal will immediately terminate the parent process and all children when in the "graceful" state. However as the `PidFile` will have been removed, you will not be able to use `apachectl` or `httpd` to send this signal.

> The `graceful-stop` signal allows you to run multiple identically configured instances of `httpd` at the same time. This is a powerful feature when performing graceful upgrades of httpd, however it can also cause deadlocks and race conditions with some configurations.
>
> Care has been taken to ensure that on-disk files such as lock files (`Mutex`) and Unix socket files (`ScriptSock`) contain the server PID, and should coexist without problem. However, if a configuration directive, third-party module or persistent CGI utilises any other on-disk lock or state files, care should be taken to ensure that multiple running instances of `httpd` do not clobber each other's files.
>
> You should also be wary of other potential race conditions, such as using `rotatelogs` style piped logging. Multiple running instances of `rotatelogs` attempting to rotate the same logfiles at the same time may destroy each other's logfiles.

模块索引

下面是 Apache HTTP 服务器发行版中的所有模块列表。参见按照字母顺序罗列的所有 Apache HTTP 服务器指令。

核心特性与多处理模块(MPM)

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)

event
A variant of the `worker` MPM with the goal of consuming threads only for connections with active processing

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

mpmt_os2
Hybrid multi-process, multi-threaded MPM for OS/2

prefork
Implements a non-threaded, pre-forking web server

mpm_winnt
Multi-Processing Module optimized for Windows NT.

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

其它模块

A | B | C | D | E | F | H | I | J | L | M | N | P | R | S | U | V | W | X

mod_access_compat
Group authorizations based on host (name or IP address)

mod_actions
Execute 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_allowhandlers
Easily restrict what HTTP handlers can be used on the server

mod_allowmethods
Easily restrict what HTTP methods can be used on the server

mod_asis
Sends files that contain their own HTTP headers

mod_auth_basic
Basic HTTP authentication

mod_auth_digest
User authentication using MD5 Digest Authentication

mod_auth_form
Form authentication

mod_authn_anon
Allows "anonymous" user access to authenticated areas

mod_authn_core
Core Authentication

mod_authn_dbd
User authentication using an SQL database

mod_authn_dbm
User authentication using DBM files

mod_authn_file
User authentication using text files

mod_authn_socache
Manages a cache of authentication credentials to relieve the load on backends

mod_authnz_fcgi
Allows a FastCGI authorizer application to handle Apache httpd authentication and authorization

mod_authnz_ldap
Allows an LDAP directory to be used to store the database for HTTP Basic authentication.

mod_authz_core
Core Authorization

mod_authz_dbd
Group Authorization and Login using SQL

mod_authz_dbm
Group authorization using DBM files

mod_authz_groupfile
Group authorization using plaintext files

mod_authz_host
Group authorizations based on host (name or IP address)

mod_authz_owner
Authorization based on file ownership

mod_authz_user
User Authorization

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

mod_brotli
Compress content via Brotli before it is delivered to the client

mod_buffer
Support for request buffering

mod_cache
RFC 2616 compliant HTTP caching filter.

mod_cache_disk
Disk based storage module for the HTTP caching filter.

mod_cache_socache
Shared object cache (socache) based storage module for the HTTP caching filter.

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_crypto
Support for symmetrical encryption and decryption

mod_data
Convert response body into an RFC2397 data URL

mod_dav
Distributed Authoring and Versioning (WebDAV) functionality

mod_dav_fs
Filesystem provider for `mod_dav`

mod_dav_lock
Generic locking module for `mod_dav`

mod_dbd
Manages SQL database connections

mod_deflate
Compress content before it is delivered to the client

mod_dialup
Send static content at a bandwidth rate limit, defined by the various old modem standards

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

mod_dumpio
Dumps all I/O to error log as desired.

mod_echo
A simple echo server to illustrate protocol modules

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

mod_example_hooks
Illustrates the Apache module API

mod_expires
Generation of `Expires` and `Cache-Control` 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_filter
Context-sensitive smart filter configuration module

mod_firehose
Multiplexes all I/O to a given file or pipe.

mod_headers
Customization of HTTP request and response headers

mod_heartbeat
Sends messages with server status to frontend proxy

mod_heartmonitor
Centralized monitor for mod_heartbeat origin servers

mod_http2
Support for the HTTP/2 transport layer

mod_ident
RFC 1413 ident lookups

mod_imagemap

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_journald
Provides "journald" ErrorLog provider

mod_lbmethod_bybusyness
Pending Request Counting load balancer scheduler algorithm for mod_proxy_balancer

mod_lbmethod_byrequests
Request Counting load balancer scheduler algorithm for mod_proxy_balancer

mod_lbmethod_bytraffic
Weighted Traffic Counting load balancer scheduler algorithm for mod_proxy_balancer

mod_lbmethod_heartbeat
Heartbeat Traffic Counting load balancer scheduler algorithm for mod_proxy_balancer

mod_ldap
LDAP connection pooling and result caching services for use by other LDAP modules

mod_log_config
Logging of the requests made to the server

mod_log_debug
Additional configurable debug logging

mod_log_forensic
Forensic Logging of the requests made to the server

mod_logio
Logging of input and output bytes per request

mod_lua
Provides Lua hooks into various portions of the httpd request processing

mod_macro
Provides macros within apache httpd runtime configuration files

mod_md
Managing domains across virtual hosts, certificate provisioning via the ACME protocol

mod_mime
Associates the requested filename's extensions 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_nw_ssl
Enable SSL encryption for NetWare

mod_policy
HTTP protocol compliance enforcement.

mod_privileges

Support for Solaris privileges and for running virtual hosts under different user IDs.

mod_proxy
Multi-protocol proxy/gateway server

mod_proxy_ajp
AJP support module for `mod_proxy`

mod_proxy_balancer
`mod_proxy` extension for load balancing

mod_proxy_connect
`mod_proxy` extension for `CONNECT` request handling

mod_proxy_express
Dynamic mass reverse proxy extension for `mod_proxy`

mod_proxy_fcgi
FastCGI support module for `mod_proxy`

mod_proxy_fdpass
fdpass external process support module for `mod_proxy`

mod_proxy_ftp
FTP support module for `mod_proxy`

mod_proxy_hcheck
Dynamic health check of Balancer members (workers) for `mod_proxy`

mod_proxy_html
Rewrite HTML links in to ensure they are addressable from Clients' networks in a proxy context.

mod_proxy_http
HTTP support module for `mod_proxy`

mod_proxy_http2
HTTP/2 support module for `mod_proxy`

mod_proxy_scgi
SCGI gateway module for `mod_proxy`

mod_proxy_uwsgi
UWSGI gateway module for `mod_proxy`

mod_proxy_wstunnel
Websockets support module for `mod_proxy`

mod_ratelimit
Bandwidth Rate Limiting for Clients

mod_reflector
Reflect a request body as a response via the output filter stack.

mod_remoteip
Replaces the original client IP address for the connection with the useragent IP address list presented by a proxies or a load balancer via the request headers.

mod_reqtimeout
Set timeout and minimum data rate for receiving requests

mod_request
Filters to handle and make available HTTP request bodies

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

mod_sed
Filter Input (request) and Output (response) content using `sed` syntax

mod_session
Session support

mod_session_cookie

Cookie based session support

mod_session_crypto
Session encryption support

mod_session_dbd
DBD/SQL based session support

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

mod_slotmem_plain
Slot-based shared memory provider.

mod_slotmem_shm
Slot-based shared memory provider.

mod_so
Loading of executable code and modules into the server at start-up or restart time

mod_socache_dbm
DBM based shared object cache provider.

mod_socache_dc
Distcache based shared object cache provider.

mod_socache_memcache
Memcache based shared object cache provider.

mod_socache_redis
Redis based shared object cache provider.

mod_socache_shmcb
shmcb based shared object cache provider.

mod_speling
Attempts to correct mistaken URLs by ignoring capitalization, or attempting to correct various minor misspellings.

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

mod_ssl_ct
Implementation of Certificate Transparency (RFC 6962)

mod_status
Provides information on server activity and performance

mod_substitute
Perform search and replace operations on response bodies

mod_suexec
Allows CGI scripts to run as a specified user and Group

mod_syslog
Provides "syslog" ErrorLog provider

mod_systemd
Provides better support for systemd integration

mod_unique_id
Provides an environment variable with a unique identifier for each request

mod_unixd
Basic (required) security for Unix-family platforms.

mod_userdir
User-specific directories

mod_usertrack
Clickstream logging of user activity on a site

mod_version
Version dependent configuration

mod_vhost_alias
: Provides for dynamically configured mass virtual hosting

mod_watchdog
: provides infrastructure for other modules to periodically run tasks

mod_xml2enc
: Enhanced charset/internationalisation support for libxml2-based filter modules

多处理模块(MPM)

此翻译可能过期。要了解最近的更改，请阅读英文版。

本文档介绍了什么是多处理模块，以及 Apache HTTP 服务器如何使用它们。

介绍

Apache HTTP 服务器被设计为一个功能强大，并且灵活的 web 服务器， 可以在很多平台与环境中工作。不同平台和不同的环境往往需要不同 的特性，或可能以不同的方式实现相同的特性最有效率。Apache httpd 通过模块化的设计来适应各种环境。这种设计允许网站管理员通过在 编译时或运行时，选择哪些模块将会加载在服务器中，来选择服务器特性。

Apache HTTP 服务器 2.0 扩展此模块化设计到最基本的 web 服务器功能。 它提供了可以选择的多处理模块(MPM)，用来绑定到网络端口上，接受请求， 以及调度子进程处理请求。

扩展到这一级别的服务器模块化设计，带来两个重要的好处:

- Apache httpd 能更优雅，更高效率的支持不同的平台。尤其是 Apache httpd 的 Windows 版本现在更有效率了，因为 mpm_winnt 能使用原生网络特性取代在 Apache httpd 1.3 中使用的 POSIX 层。它也可以扩展到其它平台 来使用专用的 MPM。
- Apache httpd 能更好的为有特殊要求的站点定制。例如，要求 更高伸缩性的站点可以选择使用线程的 MPM，即 worker 或 event； 需要可靠性或者与旧软件兼容的站点可以使用 prefork。

在用户看来，MPM 很像其它 Apache httpd 模块。主要是区别是，在任何时间，必须有一个，而且只有一个 MPM 加载到服务器中。可用的 MPM 列表位于 模块索引页面。

默认 MPM

下表列出了不同系统的默认 MPM。如果你不在编译时选择，那么它就是你将要使用的 MPM。

Netware	mpm_netware
OS/2	mpmt_os2
Unix	prefork，worker 或 event，取决于平台特性
Windows	mpm_winnt

构建 MPM 为静态模块

在全部平台中，MPM 都可以构建为静态模块。在构建时选择一种 MPM，链接到服务器中。如果要改变 MPM，必须重新构建。

为了使用指定的 MPM，请在执行 configure 脚本 时，使用参数 `--with-mpm=`*NAME*。*NAME* 是指定的 MPM 名称。

编译完成后，可以使用 `./httpd -l` 来确定选择的 MPM。 此命令会列出编译到服务器程序中的所有模块，包括 MPM。

构建 MPM 为动态模块

在 Unix 或类似平台中，MPM 可以构建为动态模块，与其它动态模块一样在运行时加载。 构建 MPM 为动态模块允许通过修改 LoadModule 指令内容来改变 MPM，而不用重新构建服务器程序。

在执行 configure 脚本时，使用 --enable-mpms-shared 选项可以启用此特性。 当给出的参数为 all 时，所有此平台支持的 MPM 模块都会被安装。还可以在参数中给出模块列表。

默认 MPM，可以自动选择或者在执行 configure 脚本时通过 --with-mpm 选项来指定，然后出现在生成的服务器配置文件中。 编辑 LoadModule 指令内容可以选择不同的 MPM。

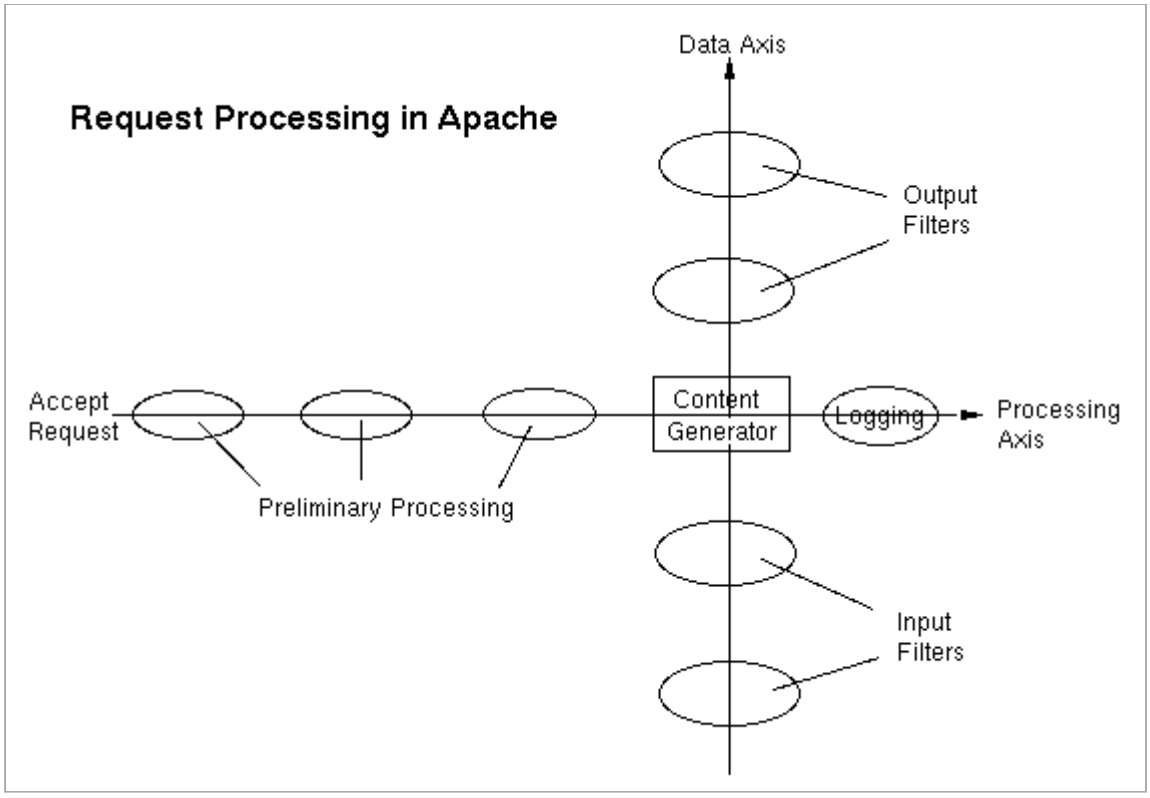
Filters

This document describes the use of filters in Apache.

Filtering in Apache 2

Related Modules	Related Directives
mod_filter	FilterChain
mod_deflate	FilterDeclare
mod_ext_filter	FilterProtocol
mod_include	FilterProvider
mod_charset_lite	AddInputFilter
mod_reflector	AddOutputFilter
mod_buffer	RemoveInputFilter
mod_data	RemoveOutputFilter
mod_ratelimit	ReflectorHeader
mod_reqtimeout	ExtFilterDefine
mod_request	ExtFilterOptions
mod_sed	SetInputFilter
mod_substitute	SetOutputFilter
mod_xml2enc	
mod_proxy_html	
mod_policy	

The Filter Chain is available in Apache 2.0 and higher, and enables applications to process incoming and outgoing data in a highly flexible and configurable manner, regardless of where the data comes from. We can pre-process incoming data, and post-process outgoing data, at will. This is basically independent of the traditional request processing phases.



Some examples of filtering in the standard Apache distribution are:

- mod_include, implements server-side includes.
- mod_ssl, implements SSL encryption (https).
- mod_deflate, implements compression/decompression on the fly.

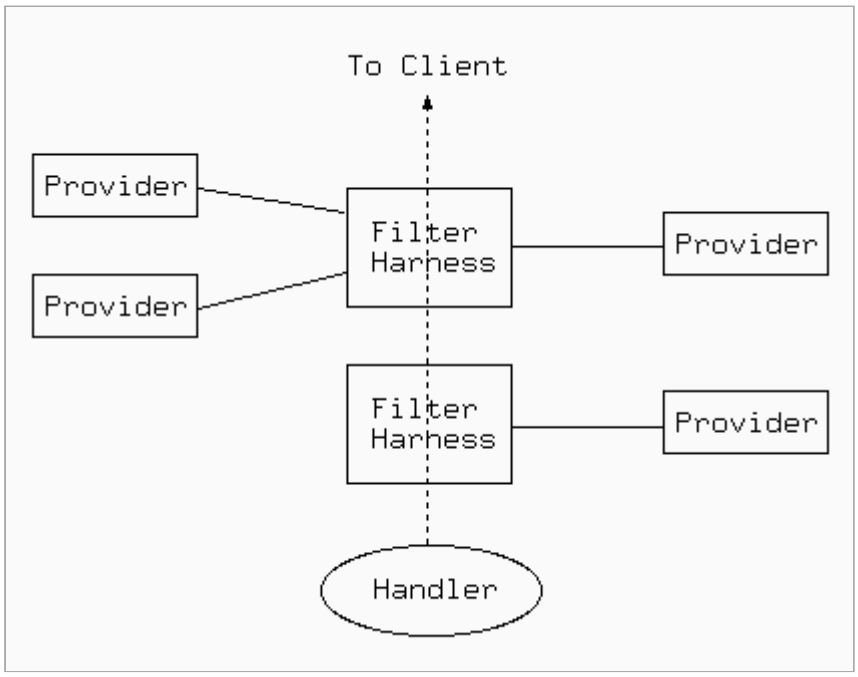
- mod_charset_lite, transcodes between different character sets.
- mod_ext_filter, runs an external program as a filter.

Apache also uses a number of filters internally to perform functions like chunking and byte-range handling.

A wider range of applications are implemented by third-party filter modules available from modules.apache.org and elsewhere. A few of these are:

- HTML and XML processing and rewriting
- XSLT transforms and XIncludes
- XML Namespace support
- File Upload handling and decoding of HTML Forms
- Image processing
- Protection of vulnerable applications such as PHP scripts
- Text search-and-replace editing

Smart Filtering



mod_filter, included in Apache 2.1 and later, enables the filter chain to be configured dynamically at run time. So for example you can set up a proxy to rewrite HTML with an HTML filter and JPEG images with a completely separate filter, despite the proxy having no prior information about what the origin server will send. This works by using a filter harness, that dispatches to different providers according to the actual contents at runtime. Any filter may be either inserted directly in the chain and run unconditionally, or used as a provider and inserted dynamically. For example,

- an HTML processing filter will only run if the content is text/html or application/xhtml+xml
- A compression filter will only run if the input is a compressible type and not already compressed
- A charset conversion filter will be inserted if a text document is not already in the desired charset

Exposing Filters as an HTTP Service

Filters can be used to process content originating from the client in addition to processing content originating on the server using the mod_reflector module.

mod_reflector accepts POST requests from clients, and reflects the content request body received within the POST request back in the response, passing through the output filter stack on the way back to the client.

This technique can be used as an alternative to a web service running within an application server stack, where an output filter provides the transformation required on the request body. For example, the mod_deflate module might be used to provide a general compression service, or an image transformation filter might be turned into an image transformation service.

Using Filters

There are two ways to use filtering: Simple and Dynamic. In general, you should use one or the other; mixing them can have unexpected consequences (although simple Input filtering can be mixed freely with either simple or dynamic Output filtering).

The Simple Way is the only way to configure input filters, and is sufficient for output filters where you need a static filter chain. Relevant directives are SetInputFilter, SetOutputFilter, AddInputFilter, AddOutputFilter, RemoveInputFilter, and RemoveOutputFilter.

The Dynamic Way enables both static and flexible, dynamic configuration of output filters, as discussed in the mod_filter page. Relevant directives are FilterChain, FilterDeclare, and FilterProvider.

One further directive AddOutputFilterByType is still supported, but deprecated. Use dynamic configuration instead.

Apache 的处理器

此翻译可能过期。要了解最近的更改，请阅读英文版。

本页描述 Apache 处理器的用法。

什么是处理器

相关模块	相关指令
mod_actions	Action
mod_asis	AddHandler
mod_cgi	RemoveHandler
mod_imagemap	SetHandler
mod_info	
mod_mime	
mod_negotiation	
mod_status	

“处理器”是当文件被调用时，Apache 要执行的动作的内部表示形式。 一般来说，每个文件都有基于其文件类型的隐式处理器。通常的文件会被 服务器简单处理，但是某些文件类型会被分别“处理”。

处理器也可以被基于扩展名或位置来明确配置。它们都很有用，这不仅 因为它是优雅的方案，而且还允许类型**与**处理器关联到文件 (参见文件与多个扩展名)。

处理器可以编译到服务器中，或者包含在模块中，它们还可以被 Action 指令增加。标准发行版中内置的处理器有:

- **default-handler**: 使用 default_handler() 发送文件，它是用来处理静态内容的处理器(核心)。
- **send-as-is**: 直接发送，不增加 HTTP 头(mod_asis)。
- **cgi-script**: 按 CGI 脚本处理(mod_cgi)。
- **imap-file**: 按 imagemap 规则处理(mod_imagemap)。
- **server-info**: 取得服务器配置信息(mod_info)。
- **server-status**: 取得服务器状态报告(mod_status)。
- **type-map**: 用于内容协商，按类型映射文件处理(mod_negotiation)。

例子

使用 CGI 脚本修改静态内容

下面的指令将会使具有html扩展名的文件，触发 CGI 脚本footer.pl的执行。

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

于是 CGI 负责发送请求的文档(PATH_TRANSLATED 环境变量指向它)，按照需要作出 and making whatever modifications or additions are desired.

含有 HTTP 头的文件

下面的指令会启用 send-as-is 处理器，用于包含自己的 HTTP 的文件。不管什么扩展名， 所有位于 /web/htdocs/asis/ 目录的文件会被 send-as-is 处理器处理。

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

对程序员的说明

为了实现处理器特性，增加了需要使用的 Apache API。 特别的，结构 `request_rec` 增加了新成员:

```
char *handler
```

如果你想要模块实现处理器，只需要在在处理请求，调用 `invoke_handler` 之前，将 `r->handler` 指向处理器名称。处理器的实现与以前一样，只是用处理器名称取代了内容类型。 虽然不是必要，处理器的命名约定是使用破折号分割的单词，没有斜杠，从而不侵入媒体类型名称空间。

Expressions in Apache HTTP Server

Historically, there are several syntax variants for expressions used to express a condition in the different modules of the Apache HTTP Server. There is some ongoing effort to only use a single variant, called *ap_expr*, for all configuration directives. This document describes the *ap_expr* expression parser.

The *ap_expr* expression is intended to replace most other expression variants in HTTPD. For example, the deprecated `SSLRequire` expressions can be replaced by Require expr.

Grammar in Backus-Naur Form notation

Backus-Naur Form (BNF) is a notation technique for context-free grammars, often used to describe the syntax of languages used in computing. In most cases, expressions are used to express boolean values. For these, the starting point in the BNF is `cond`. Directives like `ErrorDocument`, `Require`, `AuthName`, `Redirect`, `Header`, `CryptoKey` or `LogMessage` accept expressions that evaluate to a string value. For those, the starting point in the BNF is `string`.

```
expr        ::= cond
              | string

string      ::= substring
              | string substring

cond        ::= "true"
              | "false"
              | "!" cond
              | cond "&&" cond
              | cond "||" cond
              | comp
              | "(" cond ")"

comp        ::= stringcomp
              | integercomp
              | unaryop word
              | word binaryop word
              | word "in" listfunc
              | word "=~" regex
              | word "!~" regex
              | word "in" "{" list "}"


stringcomp  ::= word "==" word
              | word "!=" word
              | word "<"  word
              | word "<=" word
              | word ">"  word
              | word ">=" word

integercomp ::= word "-eq" word | word "eq" word
              | word "-ne" word | word "ne" word
              | word "-lt" word | word "lt" word
              | word "-le" word | word "le" word
              | word "-gt" word | word "gt" word
              | word "-ge" word | word "ge" word

word        ::= digits
              | "'" string "'"
              | '"' string '"'
              | word "." word
              | variable
              | sub
              | join
              | function
              | "(" word ")"

list        ::= split
              | listfunc
              | "{" words "}"
              | "(" list ")"

substring   ::= cstring
              | variable

variable    ::= "%{" varname "}"
              | "%{" funcname ":" funcargs "}"
              | "%{:" word ":}"
              | "%{:" cond ":}"
              | rebackref

sub         ::= "sub" ["("] regsub "," word [")"]

join        ::= "join" ["("] list [")"]
              | "join" ["("] list "," word [")"]

split       ::= "split" ["("] regany "," list [")"]
```

```
                | "split" ["("] regany "," word [")"]

    function    ::= funcname "(" words ")"

    listfunc    ::= listfuncname "(" words ")"

    words       ::= word
                | word "," list

    regex       ::= "/" regpattern "/" [regflags]
                | "m" regsep regpattern regsep [regflags]

    regsub      ::= "s" regsep regpattern regsep string regsep [regflags]

    regany      ::= regex | regsub

    regsep      ::= "/" | "#" | "$" | "%" | "^" | "|" | "?" | "!" | "'" | '"' | "," | ";" | ":" | "." | "_" | "-"

    regflags    ::= 1*("i" | "s" | "m" | "g")
    regpattern  ::= cstring ; except enclosing regsep

    rebackref   ::= "$" DIGIT

    digits      ::= 1*(DIGIT)
    cstring     ::= 0*(TEXT)

    TEXT        ::= <any OCTET except CTLs>
    DIGIT       ::= <any US-ASCII digit "0".."9">
```

Variables

The expression parser provides a number of variables of the form `%{HTTP_HOST}`. Note that the value of a variable may depend on the phase of the request processing in which it is evaluated. For example, an expression used in an `<If >` directive is evaluated before authentication is done. Therefore, `%{REMOTE_USER}` will not be set in this case.

The following variables provide the values of the named HTTP request headers. The values of other headers can be obtained with the `req` function. Using these variables may cause the header name to be added to the Vary header of the HTTP response, except where otherwise noted for the directive accepting the expression. The `req_novary` function may be used to circumvent this behavior.

Name
HTTP_ACCEPT
HTTP_COOKIE
HTTP_FORWARDED
HTTP_HOST
HTTP_PROXY_CONNECTION
HTTP_REFERER
HTTP_USER_AGENT

Other request related variables

Name	Description
REQUEST_METHOD	The HTTP method of the incoming request (e.g. GET)
REQUEST_SCHEME	The scheme part of the request's URI
REQUEST_URI	The path part of the request's URI
DOCUMENT_URI	Same as REQUEST_URI
REQUEST_FILENAME	The full local filesystem path to the file or script matching the request, if this has already been determined by the server at the time REQUEST_FILENAME is referenced. Otherwise, such as when used in virtual host context, the same value as REQUEST_URI
SCRIPT_FILENAME	Same as REQUEST_FILENAME
LAST_MODIFIED	The date and time of last modification of the file in the format 20101231235959, if this has already been determined by the server at the time LAST_MODIFIED is referenced.
SCRIPT_USER	The user name of the owner of the script.
SCRIPT_GROUP	The group name of the group of the script.
PATH_INFO	The trailing path name information, see AcceptPathInfo
QUERY_STRING	The query string of the current request
IS_SUBREQ	"true" if the current request is a subrequest, "false" otherwise
THE_REQUEST	The complete request line (e.g., "GET /index.html HTTP/1.1")
REMOTE_ADDR	The IP address of the remote host
REMOTE_PORT	The port of the remote host (2.4.26 and later)
REMOTE_HOST	The host name of the remote host
REMOTE_USER	The name of the authenticated user, if any (not available during <If>)
REMOTE_IDENT	The user name set by mod_ident
SERVER_NAME	The ServerName of the current vhost
SERVER_PORT	The server port of the current vhost, see ServerName

SERVER_ADMIN	The ServerAdmin of the current vhost
SERVER_PROTOCOL	The protocol used by the request (e.g. HTTP/1.1). In some types of internal subrequests, this variable has the value INCLUDED.
SERVER_PROTOCOL_VERSION	A number that encodes the HTTP version of the request: 1000 * major + minor. For example, 1001 corresponds to HTTP/1.1 and 9 corresponds to HTTP/0.9
SERVER_PROTOCOL_VERSION_MAJOR	The major version part of the HTTP version of the request, e.g. 1 for HTTP/1.0
SERVER_PROTOCOL_VERSION_MINOR	The minor version part of the HTTP version of the request, e.g. 0 for HTTP/1.0
DOCUMENT_ROOT	The DocumentRoot of the current vhost
AUTH_TYPE	The configured AuthType (e.g. "basic")
CONTENT_TYPE	The content type of the response (not available during <If>)
HANDLER	The name of the handler creating the response
HTTP2	"on" if the request uses http/2, "off" otherwise
HTTPS	"on" if the request uses https, "off" otherwise
IPV6	"on" if the connection uses IPv6, "off" otherwise
REQUEST_STATUS	The HTTP error status of the request (not available during <If>)
REQUEST_LOG_ID	The error log id of the request (see ErrorLogFormat)
CONN_LOG_ID	The error log id of the connection (see ErrorLogFormat)
CONN_REMOTE_ADDR	The peer IP address of the connection (see the mod_remoteip module)
CONTEXT_PREFIX	
CONTEXT_DOCUMENT_ROOT	

Misc variables

Name	Description
TIME_YEAR	The current year (e.g. 2010)
TIME_MON	The current month (01, ..., 12)
TIME_DAY	The current day of the month (01, ...)
TIME_HOUR	The hour part of the current time (00, ..., 23)
TIME_MIN	The minute part of the current time
TIME_SEC	The second part of the current time
TIME_WDAY	The day of the week (starting with 0 for Sunday)
TIME	The date and time in the format 20101231235959
SERVER_SOFTWARE	The server version string
API_VERSION	The date of the API version (module magic number)

Some modules register additional variables, see e.g. mod_ssl.

Any variable can be embedded in a *string*, both in quoted strings from boolean expressions but also in string expressions, resulting in the concatenation of the constant and dynamic parts as expected.

There exists another form of variables (temporaries) expressed like `%{:word:}` and which allow embedding of the more powerful *word* syntax (and constructs) in both type of expressions, without colliding with the constant part of such strings. They are mainly useful in string expressions though, since the *word* is directly available in boolean expressions already. By using this form of variables, one can evaluate regexes, substitutions, join and/or split strings and lists in the scope of string expressions, hence construct complex strings dynamically.

Binary operators

With the exception of some built-in comparison operators, binary operators have the form "`-[a-zA-Z][a-zA-Z0-9_]+`", i.e. a minus and at least two characters. The name is not case sensitive. Modules may register additional binary operators.

Comparison operators

Name	Alternative	Description
==	=	String equality
!=		String inequality
<		String less than
<=		String less than or equal
>		String greater than
>=		String greater than or equal
=~		String matches the regular expression
!~		String does not match the regular expression
-eq	eq	Integer equality
-ne	ne	Integer inequality
-lt	lt	Integer less than
-le	le	Integer less than or equal
-gt	gt	Integer greater than

-ge	ge	Integer greater than or equal

Other binary operators

Name	Description
-ipmatch	IP address matches address/netmask
-strmatch	left string matches pattern given by right string (containing wildcards *, ?, [])
-strcmatch	same as -strmatch, but case insensitive
-fnmatch	same as -strmatch, but slashes are not matched by wildcards

Unary operators

Unary operators take one argument and have the form "-[a-zA-Z]", i.e. a minus and one character. The name *is* case sensitive. Modules may register additional unary operators.

Name	Description	Restricted
-d	The argument is treated as a filename. True if the file exists and is a directory	yes
-e	The argument is treated as a filename. True if the file (or dir or special) exists	yes
-f	The argument is treated as a filename. True if the file exists and is regular file	yes
-s	The argument is treated as a filename. True if the file exists and is not empty	yes
-L	The argument is treated as a filename. True if the file exists and is symlink	yes
-h	The argument is treated as a filename. True if the file exists and is symlink (same as -L)	yes
-F	True if string is a valid file, accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to do the check, so use it with care - it can impact your server's performance!	
-U	True if string is a valid URL, accessible via all the server's currently-configured access controls for that path. This uses an internal subrequest to do the check, so use it with care - it can impact your server's performance!	
-A	Alias for -U	
-n	True if string is not empty	
-z	True if string is empty	
-T	False if string is empty, "0", "off", "false", or "no" (case insensitive). True otherwise.	
-R	Same as "%{REMOTE_ADDR} -ipmatch ...", but more efficient	

The operators marked as "restricted" are not available in some modules like mod_include.

Functions

Normal string-valued functions take one string as argument and return a string. Functions names are not case sensitive. Modules may register additional functions.

Name	Description	Special notes
req, http	Get HTTP request header; header names may be added to the Vary header, see below	
req_novary	Same as req, but header names will not be added to the Vary header	
resp	Get HTTP response header (most response headers will not yet be set during <If>)	
reqenv	Lookup request environment variable (as a shortcut, v can also be used to access variables).	ordering
osenv	Lookup operating system environment variable	
note	Lookup request note	ordering
env	Return first match of note, reqenv, osenv	ordering
tolower	Convert string to lower case	
toupper	Convert string to upper case	
escape	Escape special characters in %hex encoding	
unescape	Unescape %hex encoded string, leaving encoded slashes alone; return empty string if %00 is found	
base64	Encode the string using base64 encoding	
unbase64	Decode base64 encoded string, return truncated string if 0x00 is found	
md5	Hash the string using MD5, then encode the hash with hexadecimal encoding	
sha1	Hash the string using SHA1, then encode the hash with hexadecimal encoding	
file	Read contents from a file (including line endings, when present)	restricted
filemod	Return last modification time of a file (or 0 if file does not exist or is not regular file)	restricted
filesize	Return size of a file (or 0 if file does not exist or is not regular file)	restricted
ldap	Escape characters as required by LDAP distinguished name escaping (RFC4514) and LDAP filter escaping (RFC4515).	
replace	replace(string, "from", "to") replaces all occurrences of "from" in the string with "to".	

The functions marked as "restricted" in the final column are not available in some modules like mod_include.

The functions marked as "ordering" in the final column require some consideration for the ordering of different components of the server, especially when the function is used within the <If> directive which is evaluated relatively early.

Environment variable ordering

When environment variables are looked up within an <If> condition, it's important to consider how extremely early in request processing that this resolution occurs. As a guideline, any directive defined outside of virtual host context (directory, location, htaccess) is not likely to have yet had a chance to execute. SetEnvIf in virtual host scope is one directive that runs prior to this resolution

When reqenv is used outside of <If>, the resolution will generally occur later, but the exact timing depends on the directive the expression has been used within.

When the functions req or http are used, the header name will automatically be added to the Vary header of the HTTP response, except where otherwise noted for the directive accepting the expression. The req_novary function can be used to prevent names from being added to the Vary header.

In addition to string-valued functions, there are also list-valued functions which take one string as argument and return a list, i.e. a list of strings. The list can be used with the special -in operator. Functions names are not case sensitive. Modules may register additional functions.

There are no built-in list-valued functions. mod_ssl provides PeerExtList. See the description of SSLRequire for details (but PeerExtList is also usable outside of SSLRequire).

Other

Name	Alternative	Description
-in	in	string contained in list
/regexp/	m#regexp#	Regular expression (the second form allows different delimiters than /)
/regexp/i	m#regexp#i	Case insensitive regular expression
$0 ... $9		Regular expression backreferences

Regular expression backreferences

The strings $0 ... $9 allow to reference the capture groups from a previously executed, successfully matching regular expressions. They can normally only be used in the same expression as the matching regex, but some modules allow special uses.

Comparison with SSLRequire

The *ap_expr* syntax is mostly a superset of the syntax of the deprecated SSLRequire directive. The differences are described in SSLRequire's documentation.

Version History

The req_novary function is available for versions 2.4.4 and later.

The SERVER_PROTOCOL_VERSION, SERVER_PROTOCOL_VERSION_MAJOR and SERVER_PROTOCOL_VERSION_MINOR variables are available for versions 2.5.0 and later.

Example expressions

The following examples show how expressions might be used to evaluate requests:

```
# Compare the host name to example.com and redirect to www.example.com if it matc
<If "%{HTTP_HOST} == 'example.com'">
    Redirect permanent "/" "http://www.example.com/"
</If>

# Force text/plain if requesting a file with the query string contains 'forcetext
<If "%{QUERY_STRING} =~ /forcetext/">
    ForceType text/plain
</If>

# Only allow access to this content during business hours
<Directory "/foo/bar/business">
    Require expr %{TIME_HOUR} -gt 9 && %{TIME_HOUR} -lt 17
</Directory>

# Check a HTTP header for a list of values
<If "%{HTTP:X-example-header} in { 'foo', 'bar', 'baz' }">
    Header set matched true
</If>

# Check an environment variable for a regular expression, negated.
<If "! reqenv('REDIRECT_FOO') =~ /bar/">
    Header set matched true
</If>

# Check result of URI mapping by running in Directory context with -f
<Directory "/var/www">
    AddEncoding x-gzip gz
<If "-f '%{REQUEST_FILENAME}.unzipme' && ! %{HTTP:Accept-Encoding} =~ /gzip/">
      SetOutputFilter INFLATE
</If>
</Directory>
```

```
# Check against the client IP
<If "-R '192.168.1.0/24'">
    Header set matched true
</If>

# Function examples in boolean context
<If "md5('foo') == 'acbd18db4cc2f85cedef654fccc4a4d8'">
  Header set checksum-matched true
</If>
<If "md5('foo') == replace('md5:XXXd18db4cc2f85cedef654fccc4a4d8', 'md5:XXX', 'ac
  Header set checksum-matched-2 true
</If>

# Function example in string context
Header set foo-checksum "expr=%{md5:foo}"

# This delays the evaluation of the condition clause compared to <If>
Header always set CustomHeader my-value "expr=%{REQUEST_URI} =~ m#^/special_path\

# Add a header to forward client's certificate SAN to some backend
RequestHeader set X-Client-SAN "expr=%{:join PeerExtList('subjectAltName'):}"

# Require that the remote IP be in the client's certificate SAN
Require expr %{REMOTE_ADDR} -in split s/.*?IP Address:([^,]+)/$1/, PeerExtList('s
# or alternatively:
Require expr "IP Address:%{REMOTE_ADDR}" -in split/, /, join PeerExtList('subject

# Conditional logging
CustomLog logs/access-errors.log common "expr=%{REQUEST_STATUS} >= 400"
CustomLog logs/access-errors-specific.log common "expr=%{REQUEST_STATUS} -in {'40
```

Apache HTTP 服务器与支持程序

此翻译可能过期。要了解最近的更改，请阅读英文版。

本页描述了 Apache HTTP 服务器包含的所有可执行程序。

索引

Glossary

This glossary defines some of the common terminology related to Apache in particular, and web serving in general. More information on each concept is provided in the links.

Definitions

Access Control
The restriction of access to network realms. In an Apache context usually the restriction of access to certain *URLs*.
See: Authentication, Authorization, and Access Control

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*.

APache eXtension Tool (apxs)
A perl script that aids in compiling module sources into Dynamic Shared Objects (DSOs) and helps install them in the Apache Web server.
See: Manual Page: apxs

Apache Portable Runtime (APR)
A set of libraries providing many of the basic interfaces between the server and the operating system. APR is developed parallel to the Apache HTTP Server as an independent project.
See: Apache Portable Runtime Project

Authentication
The positive identification of a network entity such as a server, a client, or a user.
See: Authentication, Authorization, and Access Control

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 Certification 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.
See: SSL/TLS Encryption

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.
See: SSL/TLS Encryption

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.
See: SSL/TLS Encryption

Cipher
An algorithm or system for data encryption. Examples are DES, IDEA, RC4, etc.
See: SSL/TLS Encryption

Ciphertext
The result after Plaintext is passed through a Cipher.
See: SSL/TLS Encryption

Common Gateway Interface (CGI)
A standard definition for an interface between a web server and an external program that allows the external program to service requests. There is an Informational RFC which covers the specifics.
See: Dynamic Content with CGI

Configuration Directive
See: Directive

Configuration File

A text file containing Directives that control the configuration of Apache.
See: Configuration Files

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

Context
An area in the configuration files where certain types of directives are allowed.
See: Terms Used to Describe Apache Directives

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*.
See: SSL/TLS Encryption

Directive
A configuration command that controls one or more aspects of Apache's behavior. Directives are placed in the Configuration File
See: Directive Index

Dynamic Shared Object (DSO)
Modules compiled separately from the Apache `httpd` binary that can be loaded on-demand.
See: Dynamic Shared Object Support

Environment Variable (env-variable)
Named variables managed by the operating system shell and used to store information and communicate between programs. Apache also contains internal variables that are referred to as environment variables, but are stored in internal Apache structures, rather than in the shell environment.
See: Environment Variables in Apache

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.
See: SSL/TLS Encryption

Filter
A process that is applied to data that is sent or received by the server. Input filters process data sent by the client to the server, while output filters process documents on the server before they are sent to the client. For example, the `INCLUDES` output filter processes documents for Server Side Includes.
See: Filters

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, `example.com` is a domain name, and `www.example.com` is a fully-qualified domain name.

Handler
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. For example, the `cgi-script` handler designates files to be processed as CGIs.
See: Apache's Handler Use

Hash
A mathematical one-way, irreversible algorithm generating a string with fixed-length from another string of any length. Different input strings will usually produce different hashes (depending on the hash function).

Header
The part of the HTTP request and response that is sent before the actual content, and that contains meta-information describing the content.

.htaccess
A configuration file that is placed inside the web tree and applies configuration directives to the directory where it is placed and all sub-directories. Despite its name, this file can hold almost any type of directive, not just access-control directives.
See: Configuration Files

httpd.conf
: The main Apache configuration file. The default location is `/usr/local/apache2/conf/httpd.conf`, but it may be moved using run-time or compile-time configuration.
See: Configuration Files

HyperText Transfer Protocol (HTTP)
: The standard transmission protocol used on the World Wide Web. Apache implements version 1.1 of the protocol, referred to as HTTP/1.1 and defined by RFC 2616.

HTTPS
: The HyperText Transfer Protocol (Secure), the standard encrypted communication mechanism on the World Wide Web. This is actually just HTTP over SSL.
See: SSL/TLS Encryption

Method
: In the context of HTTP, an action to perform on a resource, specified on the request line by the client. Some of the methods available in HTTP are `GET`, `POST`, and `PUT`.

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.
See: SSL/TLS Encryption

MIME-type
: A way to describe the kind of document being transmitted. Its name comes from that fact that its format is borrowed from the Multipurpose Internet Mail Extensions. It consists of a major type and a minor type, separated by a slash. Some examples are `text/html`, `image/gif`, and `application/octet-stream`. In HTTP, the MIME-type is transmitted in the `Content-Type` header.
See: mod_mime

Module
: An independent part of a program. Much of Apache's functionality is contained in modules that you can choose to include or exclude. Modules that are compiled into the Apache `httpd` binary are called *static modules*, while modules that are stored separately and can be optionally loaded at run-time are called *dynamic modules* or DSOs. Modules that are included by default are called *base modules*. Many modules are available for Apache that are not distributed as part of the Apache HTTP Server tarball. These are referred to as *third-party modules*.
See: Module Index

Module Magic Number (MMN)
: Module Magic Number is a constant defined in the Apache source code that is associated with binary compatibility of modules. It is changed when internal Apache structures, function calls and other significant parts of API change in such a way that binary compatibility cannot be guaranteed any more. On MMN change, all third party modules have to be at least recompiled, sometimes even slightly changed in order to work with the new version of Apache.

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.
See: SSL/TLS Encryption

Plaintext
: The unencrypted text.

Private Key
: The secret key in a Public Key Cryptography system, used to decrypt incoming messages and sign outgoing ones.
See: SSL/TLS Encryption

Proxy
: An intermediate server that sits between the client and the *origin server*. It accepts requests from clients, transmits those requests on to the origin server, and then returns the response from the origin server to the client. If several clients request the same content, the proxy can deliver that content from its cache, rather than requesting it from the origin server each time, thereby reducing response time.
See: mod_proxy

Public Key
: The publicly available key in a Public Key Cryptography system, used to encrypt messages bound for its owner and to decrypt signatures made by its owner.
See: SSL/TLS Encryption

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 Cryptography.
See: SSL/TLS Encryption

Regular Expression (Regex)
: A way of describing a pattern in text - for example, "all the words that begin with the letter A" or "every 10-digit phone number" or even "Every sentence with two commas in it, and no capital letter Q". Regular expressions are useful in Apache because they let you apply certain attributes against collections of files or resources in very flexible ways - for example, all .gif and .jpg files under any "images" directory could be written as "`/images/.*(jpg|gif)$`". In places where regular expressions are used to replace strings, the special variables $1 ... $9 contain backreferences to the grouped parts (in parentheses) of the matched expression. The special variable $0 contains a backreference to the whole matched expression. To write a literal dollar sign in a replacement string, it can be escaped with a backslash. Historically, the variable & could be used as alias for $0 in some places. This is no longer possible since version 2.3.6. Apache uses Perl Compatible Regular Expressions provided by the PCRE library. You can find more documentation about PCRE's regular expression syntax at that site, or at Wikipedia.

Reverse Proxy
: A proxy server that appears to the client as if it is an *origin server*. This is useful to hide the real origin server from the client for security reasons, or to load balance.

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.
See: SSL/TLS Encryption

Server Name Indication (SNI)
: An SSL function that allows passing the desired server hostname in the initial SSL handshake message, so that the web server can select the correct virtual host configuration to use in processing the SSL handshake. It was added to SSL starting with the TLS extensions, RFC 3546.
See: the SSL FAQ and RFC 3546

Server Side Includes (SSI)
: A technique for embedding processing directives inside HTML files.
See: Introduction to Server Side Includes

Session
: The context information of a communication in general.

SSLeay
: The original SSL/TLS implementation library developed by Eric A. Young

Subrequest
: Apache provides a subrequest API to modules that allows other filesystem or URL paths to be partially or fully evaluated by the server. Example consumers of this API are `DirectoryIndex`, `mod_autoindex`, and `mod_include`.

Symmetric Cryptography
: The study and application of *Ciphers* that use a single secret key for both encryption and decryption operations.
See: SSL/TLS Encryption

Tarball
: A package of files gathered together using the `tar` utility. Apache distributions are stored in compressed tar archives or using pkzip.

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 is nearly identical with SSL version 3.
See: SSL/TLS Encryption

Uniform Resource Locator (URL)

The name/address of a resource on the Internet. This is the common informal term for what is formally called a Uniform Resource Identifier. URLs are usually made up of a scheme, like `http` or `https`, a hostname, and a path. A URL for this page might be `http://httpd.apache.org/docs/trunk/glossary.html`.

Uniform Resource Identifier (URI)

A compact string of characters for identifying an abstract or physical resource. It is formally defined by RFC 2396. URIs used on the world-wide web are commonly referred to as URLs.

Virtual Hosting

Serving multiple websites using a single instance of Apache. *IP virtual hosting* differentiates between websites based on their IP address, while *name-based virtual hosting* uses only the name of the host and can therefore host many sites on the same IP address.
See: Apache Virtual Host documentation

X.509

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

Binding to Addresses and Ports

Configuring Apache HTTP Server to listen on specific addresses and ports.

Overview

Related Modules	Related Directives
core	<VirtualHost>
mpm_common	Listen

When httpd starts, it binds 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 may need to be told to listen on specific ports, or only on selected addresses, or a combination of both. This is often combined with the Virtual Host feature, which determines how `httpd` responds to different IP addresses, hostnames and ports.

The `Listen` directive tells the server to accept incoming requests only on the specified port(s) 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 on. 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, on all interfaces, use:

```
Listen 80
Listen 8000
```

To make the server accept connections on port 80 for one interface, and port 8000 on another, use

```
Listen 192.0.2.1:80
Listen 192.0.2.5:8000
```

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

```
Listen [2001:db8::a00:20ff:fea7:ccea]:80
```

Overlapping `Listen` directives will result in a fatal error which will prevent the server from starting up.

```
(48)Address already in use: make_sock: could not
bind to address [::]:80
```

See the discussion in the wiki for further troubleshooting tips.

Changing Listen configuration on restart

When httpd is restarted, special consideration must be made for changes to `Listen` directives. During a restart, httpd keeps ports bound (as in the original configuration) to avoid generating "Connection refused" errors for any new attempts to connect to the server. If changes are made to the set of `Listen`

directives used which conflict with the old configuration, configuration will fail and the server will terminate.

For example, changing from configuration:

```
Listen 127.0.0.1:80
```

to the following may fail, because binding to port 80 across all addresses conflicts with binding to port 80 on just 127.0.0.1.

```
Listen 80
```

To have such configuration changes take effect, it is necessary to stop and then start the server.

Special IPv6 Considerations

A growing number of platforms implement IPv6, and APR supports IPv6 on most of these platforms, allowing httpd to allocate IPv6 sockets, and to handle requests sent over IPv6.

One complicating factor for httpd administrators is whether or not an IPv6 socket can handle both IPv4 connections and IPv6 connections. Handling IPv4 connections with an IPv6 socket uses IPv4-mapped IPv6 addresses, which are allowed by default on most platforms, but are disallowed by default on FreeBSD, NetBSD, and OpenBSD, in order to match the system-wide policy on those platforms. On systems where it is disallowed by default, a special `configure` parameter can change this behavior for httpd.

On the other hand, on some platforms, such as Linux and Tru64, the **only** way to handle both IPv6 and IPv4 is to use mapped addresses. If you want `httpd` to handle IPv4 and IPv6 connections with a minimum of sockets, which requires using IPv4-mapped IPv6 addresses, specify the `--enable-v4-mapped` `configure` option.

`--enable-v4-mapped` is the default on all platforms except FreeBSD, NetBSD, and OpenBSD, so this is probably how your httpd was built.

If you want httpd to handle IPv4 connections only, regardless of what your platform and APR will support, specify an IPv4 address on all `Listen` directives, as in the following examples:

```
Listen 0.0.0.0:80
Listen 192.0.2.1:80
```

If your platform supports it and you want httpd to handle IPv4 and IPv6 connections on separate sockets (i.e., to disable IPv4-mapped addresses), specify the `--disable-v4-mapped` `configure` option. `--disable-v4-mapped` is the default on FreeBSD, NetBSD, and OpenBSD.

Specifying the protocol with Listen

The optional second *protocol* argument of `Listen` is not required for most configurations. If not specified, `https` is the default for port 443 and `http` the default for all other ports. The protocol is used to determine which module should handle a request, and to apply protocol specific optimizations with the `AcceptFilter` directive.

You only need to set the protocol if you are running on non-standard ports. For example, running an `https` site on port 8443:

```
Listen 192.170.2.1:8443 https
```

How This Works With Virtual Hosts

The `Listen` directive does not implement Virtual Hosts - it only tells the main server what addresses and ports to listen on. If no `<VirtualHost>` directives are used, the server will behave in the same way for all accepted requests. However, `<VirtualHost>` can be used to specify a different behavior for one or more of the addresses or 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 the specified address and port to set the behavior of this virtual host. Note that if the `<VirtualHost>` is set for an address and port that the server is not listening to, it cannot be accessed.

Configuration Files

This document describes the files used to configure Apache HTTP Server.

Main Configuration Files

Related Modules	Related Directives
mod_mime	<IfDefine> Include TypesConfig

Apache HTTP Server 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, and wildcards can be used to include many configuration files. Any directive may be placed in any of these configuration files. Changes to the main configuration files are only recognized by httpd when it is started or restarted.

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

httpd configuration files contain one directive per line. The backslash "\" 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 backslash and the end of the line.

Arguments to directives are separated by whitespace. If an argument contains spaces, you must enclose that argument in quotes.

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

The values of variables defined with the `Define` or of shell environment variables can be used in configuration file lines using the syntax `${VAR}`.
If "VAR" is the name of a valid variable, the value of that variable is substituted into that spot in the configuration file line, and processing continues as if that text were found directly in the configuration file.
Variables defined with `Define` take precedence over shell environment variables.
If the "VAR" variable is not defined, the characters `${VAR}` are left unchanged, and a warning is logged. If instead a default value should be substituted, the conditional form `${VAR?=some default value}` can be used. Note that a **defined** empty variable will **not** be substituted with the default value, and that an empty default value like in `${VAR?=}` is a valid substitution (which produces an empty value if "VAR" is not defined, but no warning).
Variable names may not contain colon ":" characters, to avoid clashes with `RewriteMap`'s syntax.

Only shell environment variables defined before the server is started can be used in expansions. Environment variables defined in the configuration file itself, for example with `SetEnv`, take effect too late to be used for expansions in the configuration file.

The maximum length of a line in normal configuration files, after variable substitution and joining any continued lines, is approximately 16 MiB. In .htaccess files, the maximum length is 8190 characters.

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

You can use mod_info's `-DDUMP_CONFIG` to dump the configuration with all included files and environment variables resolved and all comments and non-matching <IfDefine> and <IfModule> sections removed. However, the output does not reflect the merging or overriding that may happen for repeated directives.

Modules

Related Modules	Related Directives
mod_so	<IfModule>
	LoadModule

httpd 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 httpd. 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, httpd 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. However, <IfModule> blocks are not required, and in some cases may mask the fact that you're missing an important module.

To see which modules are currently compiled into the server, you can use the `-l` command line option. You can also see what modules are loaded dynamically using the `-M` command line option.

Scope of Directives

Related Modules	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.

httpd 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 Modules	Related Directives
	AccessFileName
	AllowOverride

httpd 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 the .htaccess tutorial.

Configuration Sections

Directives in the configuration files may apply to the entire server, or they may be restricted to apply only to particular directories, files, hosts, or URLs. This document describes how to use configuration section containers or `.htaccess` files to change the scope of other configuration directives.

Types of Configuration Section Containers

Related Modules	Related Directives
core	<Directory>
mod_version	<DirectoryMatch>
mod_proxy	<Files>
	<FilesMatch>
	<If>
	<IfDefine>
	<IfModule>
	<IfVersion>
	<Location>
	<LocationMatch>
	<MDomainSet>
	<Proxy>
	<ProxyMatch>
	<VirtualHost>

There are two basic types of containers. Most containers are evaluated for each request. The enclosed directives are applied only for those requests that match the containers. The <IfDefine>, <IfModule>, and <IfVersion> containers, on the other hand, are evaluated only at server startup and restart. If their conditions are true at startup, then the enclosed directives will apply to all requests. If the conditions are not true, the enclosed directives will be ignored.

The <IfDefine> directive encloses directives that will only be applied if an appropriate parameter is defined on the httpd command line, or with a Define directive. For example, with the following configuration, all requests will be redirected to another site only if the server is started using `httpd -DClosedForNow`:

```
<IfDefine ClosedForNow>
    Redirect "/" "http://otherserver.example.com/"
</IfDefine>
```

The <IfModule> directive is very similar, except it encloses directives that will only be applied if a particular module is available in the server. The module must either be statically compiled in the server, or it must be dynamically compiled and its LoadModule line must be earlier in the configuration file. This directive should only be used if you need your configuration file to work whether or not certain modules are installed. It should not be used to enclose directives that you want to work all the time, because it can suppress useful error messages about missing modules.

In the following example, the MimeMagicFile directive will be applied only if mod_mime_magic is available.

```
<IfModule mod_mime_magic.c>
    MimeMagicFile "conf/magic"
</IfModule>
```

The <IfVersion> directive is very similar to <IfDefine> and <IfModule>, except it encloses directives that will only be applied if a particular version of the server is executing. This module is designed for the use in test suites and large networks which have to deal with different httpd versions and different configurations.

```
<IfVersion >= 2.4>
    # this happens only in versions greater or
    # equal 2.4.0.
</IfVersion>
```

<IfDefine>, <IfModule>, and the <IfVersion> can apply negative conditions by preceding their test with "!". Also, these sections can be nested to achieve more complex restrictions.

Filesystem, Webspace, and Boolean Expressions

The most commonly used configuration section containers are the ones that change the configuration of particular places in the filesystem or webspace. First, it is important to understand the difference between the two. The filesystem is the view of your disks as seen by your operating system. For example, in a default install, Apache httpd resides at `/usr/local/apache2` in the Unix filesystem or `"c:/Program Files/Apache Group/Apache2"` in the Windows filesystem. (Note that forward slashes should always be used as the path separator in Apache httpd configuration files, even for Windows.) In contrast, the webspace is the view of your site as delivered by the web server and seen by the client. So the path `/dir/` in the webspace corresponds to the path `/usr/local/apache2/htdocs/dir/` in the filesystem of a default Apache httpd install on Unix. The webspace need not map directly to the filesystem, since webpages may be generated dynamically from databases or other locations.

Filesystem Containers

The <Directory> and <Files> directives, along with their regex counterparts, apply directives to parts of the filesystem. Directives enclosed in a <Directory> section apply to the named filesystem directory and all subdirectories of that directory (as well as the files in those directories). The same effect can be obtained using .htaccess files. For example, in the following configuration, directory indexes will be enabled for the `/var/web/dir1` directory and all subdirectories.

```
<Directory "/var/web/dir1">
    Options +Indexes
</Directory>
```

Directives enclosed in a <Files> section apply to any file with the specified name, regardless of what directory it lies in. So for example, the following configuration directives will, when placed in the main section of the configuration file, deny access to any file named `private.html` regardless of where it is found.

```
<Files "private.html">
    Require all denied
</Files>
```

To address files found in a particular part of the filesystem, the <Files> and <Directory> sections can be combined. For example, the following configuration will deny access to `/var/web/dir1/private.html`, `/var/web/dir1/subdir2/private.html`, `/var/web/dir1/subdir3/private.html`, and any other instance of `private.html` found under the `/var/web/dir1/` directory.

```
<Directory "/var/web/dir1">
    <Files "private.html">
        Require all denied
    </Files>
</Directory>
```

Webspace Containers

The <Location> directive and its regex counterpart, on the other hand, change the configuration for content in the webspace. For example, the following configuration prevents access to any URL-path that begins in /private. In particular, it will apply to requests for `http://yoursite.example.com/private`, `http://yoursite.example.com/private123`, and `http://yoursite.example.com/private/dir/file.html` as well as any other requests starting with the `/private` string.

```
<LocationMatch "^/private">
    Require all denied
</LocationMatch>
```

The <Location> directive need not have anything to do with the filesystem. For example, the following example shows how to map a particular URL to an internal Apache HTTP Server handler provided by mod_status. No file called `server-status` needs to exist in the filesystem.

```
<Location "/server-status">
    SetHandler server-status
</Location>
```

Overlapping Webspace

In order to have two overlapping URLs one has to consider the order in which certain sections or directives are evaluated. For <Location> this would be:

```
<Location "/foo">
</Location>
<Location "/foo/bar">
</Location>
```

<Alias>es on the other hand, are mapped vice-versa:

```
Alias "/foo/bar" "/srv/www/uncommon/bar"
Alias "/foo"     "/srv/www/common/foo"
```

The same is true for the ProxyPass directives:

```
ProxyPass "/special-area" "http://special.example.com"
ProxyPass "/" "balancer://mycluster/" stickysession=JSE
```

Wildcards and Regular Expressions

The <Directory>, <Files>, and <Location> directives can each use shell-style wildcard characters as in `fnmatch` from the C standard library. The character "*" matches any sequence of characters, "?" matches any single character, and "[*seq*]" matches any character in *seq*. The "/" character will not be matched by any wildcard; it must be specified explicitly.

If even more flexible matching is required, each container has a regular expression (regex) counterpart <DirectoryMatch>, <FilesMatch>, and <LocationMatch> that allow perl-compatible regular expressions to be used in

choosing the matches. But see the section below on configuration merging to find out how using regex sections will change how directives are applied.

A non-regex wildcard section that changes the configuration of all user directories could look as follows:

```
<Directory "/home/*/public_html">
    Options Indexes
</Directory>
```

Using regex sections, we can deny access to many types of image files at once:

```
<FilesMatch "\.(?i:gif|jpe?g|png)$">
    Require all denied
</FilesMatch>
```

Regular expressions containing **named groups and backreferences** are added to the environment with the corresponding name in uppercase. This allows elements of filename paths and URLs to be referenced from within expressions and modules like `mod_rewrite`.

```
<DirectoryMatch "^/var/www/combined/(?<SITENAME>[^/]+)'
    Require ldap-group "cn=%{env:MATCH_SITENAME},ou=com
</DirectoryMatch>
```

Boolean expressions

The `<If>` directive change the configuration depending on a condition which can be expressed by a boolean expression. For example, the following configuration denies access if the HTTP Referer header does not start with "http://www.example.com/".

```
<If "!(%{HTTP_REFERER} -strmatch 'http://www.example.co
    Require all denied
</If>
```

What to use When

Choosing between filesystem containers and webspace containers is actually quite easy. When applying directives to objects that reside in the filesystem always use `<Directory>` or `<Files>`. When applying directives to objects that do not reside in the filesystem (such as a webpage generated from a database), use `<Location>`.

It is important to never use `<Location>` when trying to restrict access to objects in the filesystem. This is because many different webspace locations (URLs) could map to the same filesystem location, allowing your restrictions to be circumvented. For example, consider the following configuration:

```
<Location "/dir/">
    Require all denied
</Location>
```

This works fine if the request is for `http://yoursite.example.com/dir/`. But what if you are on a case-insensitive filesystem? Then your restriction could be easily circumvented by requesting `http://yoursite.example.com/DIR/`. The `<Directory>` directive, in contrast, will apply to any content served from that location, regardless of how it is called. (An exception is filesystem links. The same directory can be placed in more than one part of the filesystem using symbolic links. The `<Directory>` directive will follow the symbolic link without resetting the pathname. Therefore, for the highest level of security, symbolic links should be disabled with the appropriate `Options` directive.)

If you are, perhaps, thinking that none of this applies to you because you use a case-sensitive filesystem, remember that there are many other ways to map multiple webspace locations to the same filesystem location. Therefore you should always use the filesystem containers when you can. There is, however, one exception to this rule. Putting configuration restrictions in a `<Location "/">` section is perfectly safe because this section will apply to all requests regardless of the specific URL.

Nesting of sections

Some section types can be nested inside other section types. On the one hand, `<Files>` can be used inside `<Directory>`. On the other hand, `<If>` can be used inside `<Directory>`, `<Location>`, and `<Files>` sections (but not inside another `<If>`). The regex counterparts of the named section behave identically.

Nested sections are merged after non-nested sections of the same type.

Virtual Hosts

The `<VirtualHost>` container encloses directives that apply to specific hosts. This is useful when serving multiple hosts from the same machine with a different configuration for each. For more information, see the Virtual Host Documentation.

Proxy

The `<Proxy>` and `<ProxyMatch>` containers apply enclosed configuration directives only to sites accessed through `mod_proxy`'s proxy server that match the specified URL. For example, the following configuration will allow only a subset of clients to access the `www.example.com` website using the proxy server:

```
<Proxy "http://www.example.com/*">
    Require host yournetwork.example.com
</Proxy>
```

What Directives are Allowed?

To find out what directives are allowed in what types of configuration sections, check the Context of the directive. Everything that is allowed in `<Directory>` sections is also syntactically allowed in `<DirectoryMatch>`, `<Files>`, `<FilesMatch>`, `<Location>`, `<LocationMatch>`, `<Proxy>`, and `<ProxyMatch>` sections. There are some exceptions, however:

- The `AllowOverride` directive works only in `<Directory>` sections.
- The `FollowSymLinks` and `SymLinksIfOwnerMatch` `Options` work only in `<Directory>` sections or `.htaccess` files.
- The `Options` directive cannot be used in `<Files>` and `<FilesMatch>` sections.

How the sections are merged

The configuration sections are applied in a very particular order. Since this can have important effects on how configuration directives are interpreted, it is important to understand how this works.

The order of merging is:

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

4. <Location> and <LocationMatch> done simultaneously
5. <If>

Some important remarks:

- Apart from <Directory>, within each group the sections are processed in the order they appear in the configuration files. For example, a request for */foo/bar* will match <Location "/foo/bar"> and <Location "/foo"> (group 4 in this case): both sections will be evaluated but in the order they appear in the configuration files.
- <Directory> (group 1 above) is processed in the order shortest directory component to longest. For example, <Directory "/var/web/dir"> will be processed before <Directory "/var/web/dir/subdir">.
- If multiple <Directory> sections apply to the same directory they are processed in the configuration file order.
- 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.
- When the request is served by mod_proxy, the <Proxy> container takes the place of the <Directory> container in the processing order.

> **Technical 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.

Relationship between modules and configuration sections

One question that often arises after reading how configuration sections are merged is related to how and when directives of specific modules like mod_rewrite are processed. The answer is not trivial and needs a bit of background. Each httpd module manages its own configuration, and each of its directives in httpd.conf specify one piece of configuration in a particular context. httpd does not execute a command as it is read.

At runtime, the core of httpd iterates over the defined configuration sections in the order described above to determine which ones apply to the current request. When the first section matches, it is considered the current configuration for this request. If a subsequent section matches too, then each module with a directive in either of the sections is given a chance to merge its configuration between the two sections. The result is a third configuration, and the process goes on until all the configuration sections are evaluated.

After the above step, the "real" processing of the HTTP request begins: each module has a chance to run and perform whatever tasks they like. They can retrieve their own final merged configuration from the core of the httpd to determine how they should act.

An example can help to visualize the whole process. The following configuration uses the Header directive of mod_headers to set a specific HTTP header. What value will httpd set in the CustomHeaderName header for a request to /example/index.html ?

```
<Directory "/">
    Header set CustomHeaderName one
    <FilesMatch ".*">
        Header set CustomHeaderName three
    </FilesMatch>
```

```
</Directory>

<Directory "/example">
    Header set CustomHeaderName two
</Directory>
```

- `Directory` "/" matches and an initial configuration to set the `CustomHeaderName` header with the value `one` is created.
- `Directory` "/example" matches, and since `mod_headers` specifies in its code to override in case of a merge, a new configuration is created to set the `CustomHeaderName` header with the value `two`.
- `FilesMatch` ".*" matches and another merge opportunity arises, causing the `CustomHeaderName` header to be set with the value `three`.
- Eventually during the next steps of the HTTP request processing `mod_headers` will be called and it will receive the configuration to set the `CustomHeaderName` header with the value `three`. `mod_headers` normally uses this configuration to perform its job, namely setting the foo header. This does not mean that a module can't perform a more complex action like discarding directives because not needed or deprecated, etc..

This is true for .htaccess too since they have the same priority as `Directory` in the merge order. The important concept to understand is that configuration sections like `Directory` and `FilesMatch` are not comparable to module specific directives like `Header` or `RewriteRule` because they operate on different levels.

Some useful examples

Below is an artificial example to show the order of merging. Assuming they all apply to the request, the directives in this example will be applied in the order A > B > C > D > E.

```
<Location "/">
    E
</Location>

<Files "f.html">
    D
</Files>

<VirtualHost *>
    <Directory "/a/">
        B
    </Directory>
</VirtualHost>

<DirectoryMatch "^.*b$">
    C
</DirectoryMatch>

<Directory "/a/b">
    A
</Directory>
```

For a more concrete example, consider the following. Regardless of any access restrictions placed in `<Directory>` sections, the `<Location>` section will be evaluated last and will allow unrestricted access to the server. In other words, order of merging is important, so be careful!

```
<Location "/">
    Require all granted
</Location>

# Whoops!  This <Directory> section will have no effect
<Directory "/">
```

```
    <RequireAll>
        Require all granted
        Require not host badguy.example.com
    </RequireAll>
</Directory>
```

Caching Guide

This document supplements the mod_cache, mod_cache_disk, mod_file_cache and htcacheclean reference documentation. It describes how to use the Apache HTTP Server's caching features to accelerate web and proxy serving, while avoiding common problems and misconfigurations.

Introduction

The Apache HTTP server offers a range of caching features that are designed to improve the performance of the server in various ways.

Three-state RFC2616 HTTP caching
: mod_cache and its provider modules mod_cache_disk provide intelligent, HTTP-aware caching. The content itself is stored in the cache, and mod_cache aims to honor all of the various HTTP headers and options that control the cacheability of content as described in Section 13 of RFC2616. mod_cache is aimed at both simple and complex caching configurations, where you are dealing with proxied content, dynamic local content or have a need to speed up access to local files on a potentially slow disk.

Two-state key/value shared object caching
: The shared object cache API (socache) and its provider modules provide a server wide key/value based shared object cache. These modules are designed to cache low level data such as SSL sessions and authentication credentials. Backends allow the data to be stored server wide in shared memory, or datacenter wide in a cache such as memcache or distcache.

Specialized file caching
: mod_file_cache offers the ability to pre-load files into memory on server startup, and can improve access times and save file handles on files that are accessed often, as there is no need to go to disk on each request.

To get the most from this document, you should be familiar with the basics of HTTP, and have read the Users' Guides to Mapping URLs to the Filesystem and Content negotiation.

Three-state RFC2616 HTTP caching

Related Modules	Related Directives
mod_cache	CacheEnable
mod_cache_disk	CacheDisable
	UseCanonicalName
	CacheNegotiatedDocs

The HTTP protocol contains built in support for an in-line caching mechanism described by section 13 of RFC2616, and the mod_cache module can be used to take advantage of this.

Unlike a simple two state key/value cache where the content disappears completely when no longer fresh, an HTTP cache includes a mechanism to retain stale content, and to ask the origin server whether this stale content has changed and if not, make it fresh again.

An entry in an HTTP cache exists in one of three states:

Fresh
: If the content is new enough (younger than its **freshness lifetime**), it is considered **fresh**. An HTTP cache is free to serve fresh content without making any calls to the origin server at all.

Stale

If the content is too old (older than its **freshness lifetime**), it is considered **stale**. An HTTP cache should contact the origin server and check whether the content is still fresh before serving stale content to a client. The origin server will either respond with replacement content if not still valid, or ideally, the origin server will respond with a code to tell the cache the content is still fresh, without the need to generate or send the content again. The content becomes fresh again and the cycle continues.

The HTTP protocol does allow the cache to serve stale data under certain circumstances, such as when an attempt to freshen the data with an origin server has failed with a 5xx error, or when another request is already in the process of freshening the given entry. In these cases a `Warning` header is added to the response.

Non Existent

If the cache gets full, it reserves the option to delete content from the cache to make space. Content can be deleted at any time, and can be stale or fresh. The htcacheclean tool can be run on a once off basis, or deployed as a daemon to keep the size of the cache within the given size, or the given number of inodes. The tool attempts to delete stale content before attempting to delete fresh content.

Full details of how HTTP caching works can be found in Section 13 of RFC2616.

Interaction with the Server

The `mod_cache` module hooks into the server in two possible places depending on the value of the `CacheQuickHandler` directive:

Quick handler phase

This phase happens very early on during the request processing, just after the request has been parsed. If the content is found within the cache, it is served immediately and almost all request processing is bypassed.

In this scenario, the cache behaves as if it has been "bolted on" to the front of the server.

This mode offers the best performance, as the majority of server processing is bypassed. This mode however also bypasses the authentication and authorization phases of server processing, so this mode should be chosen with care when this is important.

Requests with an "Authorization" header (for example, HTTP Basic Authentication) are neither cacheable nor served from the cache when `mod_cache` is running in this phase.

Normal handler phase

This phase happens late in the request processing, after all the request phases have completed.

In this scenario, the cache behaves as if it has been "bolted on" to the back of the server.

This mode offers the most flexibility, as the potential exists for caching to occur at a precisely controlled point in the filter chain, and cached content can be filtered or personalized before being sent to the client.

If the URL is not found within the cache, `mod_cache` will add a filter to the filter stack in order to record the response to the cache, and then stand down, allowing normal request processing to continue. If the content is determined to be cacheable, the content will be saved to the cache for future serving, otherwise the content will be ignored.

If the content found within the cache is stale, the mod_cache module converts the request into a **conditional request**. If the origin server responds with a normal response, the normal response is cached, replacing the content already cached. If the origin server responds with a 304 Not Modified response, the content is marked as fresh again, and the cached content is served by the filter instead of saving it.

Improving Cache Hits

When a virtual host is known by one of many different server aliases, ensuring that UseCanonicalName is set to On can dramatically improve the ratio of cache hits. This is because the hostname of the virtual-host serving the content is used within the cache key. With the setting set to On virtual-hosts with multiple server names or aliases will not produce differently cached entities, and instead content will be cached as per the canonical hostname.

Freshness Lifetime

Well formed content that is intended to be cached should declare an explicit freshness lifetime with the Cache-Control header's max-age or s-maxage fields, or by including an Expires header.

At the same time, the origin server defined freshness lifetime can be overridden by a client when the client presents their own Cache-Control header within the request. In this case, the lowest freshness lifetime between request and response wins.

When this freshness lifetime is missing from the request or the response, a default freshness lifetime is applied. The default freshness lifetime for cached entities is one hour, however this can be easily over-ridden by using the CacheDefaultExpire directive.

If a response does not include an Expires header but does include a Last-Modified header, mod_cache can infer a freshness lifetime based on a heuristic, which can be controlled through the use of the CacheLastModifiedFactor directive.

For local content, or for remote content that does not define its own Expires header, mod_expires may be used to fine-tune the freshness lifetime by adding max-age and Expires.

The maximum freshness lifetime may also be controlled by using the CacheMaxExpire.

A Brief Guide to Conditional Requests

When content expires from the cache and becomes stale, rather than pass on the original request, httpd will modify the request to make it conditional instead.

When an ETag header exists in the original cached response, mod_cache will add an If-None-Match header to the request to the origin server. When a Last-Modified header exists in the original cached response, mod_cache will add an If-Modified-Since header to the request to the origin server. Performing either of these actions makes the request **conditional**.

When a conditional request is received by an origin server, the origin server should check whether the ETag or the Last-Modified parameter has changed, as appropriate for the request. If not, the origin should respond with a terse "304 Not Modified" response. This signals to the cache that the stale content is still fresh should be used for subsequent requests until the content's new freshness lifetime is reached again.

If the content has changed, then the content is served as if the request were not conditional to begin with.

Conditional requests offer two benefits. Firstly, when making such a request to the origin server, if the content from the origin matches the content in the cache, this can be determined easily and without the overhead of transferring the entire resource.

Secondly, a well designed origin server will be designed in such a way that conditional requests will be significantly cheaper to produce than a full response. For static files, typically all that is involved is a call to `stat()` or similar system call, to see if the file has changed in size or modification time. As such, even local content may still be served faster from the cache if it has not changed.

Origin servers should make every effort to support conditional requests as is practical, however if conditional requests are not supported, the origin will respond as if the request was not conditional, and the cache will respond as if the content had changed and save the new content to the cache. In this case, the cache will behave like a simple two state cache, where content is effectively either fresh or deleted.

What Can be Cached?

The full definition of which responses can be cached by an HTTP cache is defined in RFC2616 Section 13.4 Response Cacheability, and can be summed up as follows:

1. Caching must be enabled for this URL. See the `CacheEnable` and `CacheDisable` directives.
2. If the response has an HTTP status code other than 200, 203, 300, 301 or 410 it must also specify an "Expires" or "Cache-Control" header.
3. The request must be a HTTP GET request.
4. If the response contains an "Authorization:" header, it must also contain an "s-maxage", "must-revalidate" or "public" option in the "Cache-Control:" header, or it won't be cached.
5. If the URL included a query string (e.g. from a HTML form GET method) it will not be cached unless the response specifies an explicit expiration by including an "Expires:" header or the max-age or s-maxage directive of the "Cache-Control:" header, as per RFC2616 sections 13.9 and 13.2.1.
6. If the response has a status of 200 (OK), the response must also include at least one of the "Etag", "Last-Modified" or the "Expires" headers, or the max-age or s-maxage directive of the "Cache-Control:" header, unless the `CacheIgnoreNoLastMod` directive has been used to require otherwise.
7. If the response includes the "private" option in a "Cache-Control:" header, it will not be stored unless the `CacheStorePrivate` has been used to require otherwise.
8. Likewise, if the response includes the "no-store" option in a "Cache-Control:" header, it will not be stored unless the `CacheStoreNoStore` has been used.
9. A response will not be stored if it includes a "Vary:" header containing the match-all "*".

What Should Not be Cached?

It should be up to the client creating the request, or the origin server constructing the response to decide whether or not the content should be cacheable or not by correctly setting the `Cache-Control` header, and `mod_cache` should be left alone to honor the wishes of the client or server as appropriate.

Content that is time sensitive, or which varies depending on the particulars of the request that are not covered by HTTP negotiation, should not be cached. This content should declare itself uncacheable using the `Cache-Control` header.

If content changes often, expressed by a freshness lifetime of minutes or seconds, the content can still be cached, however it is highly desirable that the origin server supports **conditional requests** correctly to ensure that full responses do not have to be generated on a regular basis.

Content that varies based on client provided request headers can be cached through intelligent use of the `Vary` response header.

Variable/Negotiated Content

When the origin server is designed to respond with different content based on the value of headers in the request, for example to serve multiple languages at the same URL, HTTP's caching mechanism makes it possible to cache multiple variants of the same page at the same URL.

This is done by the origin server adding a `Vary` header to indicate which headers must be taken into account by a cache when determining whether two variants are different from one another.

If for example, a response is received with a vary header such as;

```
Vary: negotiate,accept-language,accept-charset
```

mod_cache will only serve the cached content to requesters with accept-language and accept-charset headers matching those of the original request.

Multiple variants of the content can be cached side by side, mod_cache uses the `Vary` header and the corresponding values of the request headers listed by `Vary` to decide on which of many variants to return to the client.

Cache Setup Examples

Related Modules	Related Directives
mod_cache	CacheEnable
mod_cache_disk	CacheRoot
mod_cache_socache	CacheDirLevels
mod_socache_memcache	CacheDirLength
	CacheSocache

Caching to Disk

The mod_cache module relies on specific backend store implementations in order to manage the cache, and for caching to disk mod_cache_disk is provided to support this.

Typically the module will be configured as so;

```
CacheRoot   "/var/cache/apache/"
CacheEnable disk /
CacheDirLevels 2
CacheDirLength 1
```

Importantly, as the cached files are locally stored, operating system in-memory caching will typically be applied to their access also. So although the files are stored on disk, if they are frequently accessed it is likely the operating system will ensure that they are actually served from memory.

Understanding the Cache-Store

To store items in the cache, mod_cache_disk creates a 22 character hash of the URL being requested. This hash incorporates the hostname, protocol, port, path and any CGI arguments to the URL, as well as elements defined by the Vary header to ensure that multiple URLs do not collide with one another.

Each character may be any one of 64-different characters, which mean that overall there are 64^22 possible hashes. For example, a URL might be hashed to `xyTGxSMO2b68mBCykqkp1w`. This hash is used as a prefix for the naming of the files specific to that URL within the cache, however first it is split up into directories as per the `CacheDirLevels` and `CacheDirLength` directives.

`CacheDirLevels` specifies how many levels of subdirectory there should be, and `CacheDirLength` specifies how many characters should be in each directory. With the example settings given above, the hash would be turned into a filename prefix as `/var/cache/apache/x/y/TGxSMO2b68mBCykqkp1w`.

The overall aim of this technique is to reduce the number of subdirectories or files that may be in a particular directory, as most file-systems slow down as this number increases. With setting of "1" for `CacheDirLength` there can at most be 64 subdirectories at any particular level. With a setting of 2 there can be 64 * 64 subdirectories, and so on. Unless you have a good reason not to, using a setting of "1" for `CacheDirLength` is recommended.

Setting `CacheDirLevels` depends on how many files you anticipate to store in the cache. With the setting of "2" used in the above example, a grand total of 4096 subdirectories can ultimately be created. With 1 million files cached, this works out at roughly 245 cached URLs per directory.

Each URL uses at least two files in the cache-store. Typically there is a ".header" file, which includes meta-information about the URL, such as when it is due to expire and a ".data" file which is a verbatim copy of the content to be served.

In the case of a content negotiated via the "Vary" header, a ".vary" directory will be created for the URL in question. This directory will have multiple ".data" files corresponding to the differently negotiated content.

Maintaining the Disk Cache

The `mod_cache_disk` module makes no attempt to regulate the amount of disk space used by the cache, although it will gracefully stand down on any disk error and behave as if the cache was never present.

Instead, provided with httpd is the htcacheclean tool which allows you to clean the cache periodically. Determining how frequently to run htcacheclean and what target size to use for the cache is somewhat complex and trial and error may be needed to select optimal values.

htcacheclean has two modes of operation. It can be run as persistent daemon, or periodically from cron. htcacheclean can take up to an hour or more to process very large (tens of gigabytes) caches and if you are running it from cron it is recommended that you determine how long a typical run takes, to avoid running more than one instance at a time.

It is also recommended that an appropriate "nice" level is chosen for htcacheclean so that the tool does not cause excessive disk io while the server is running.

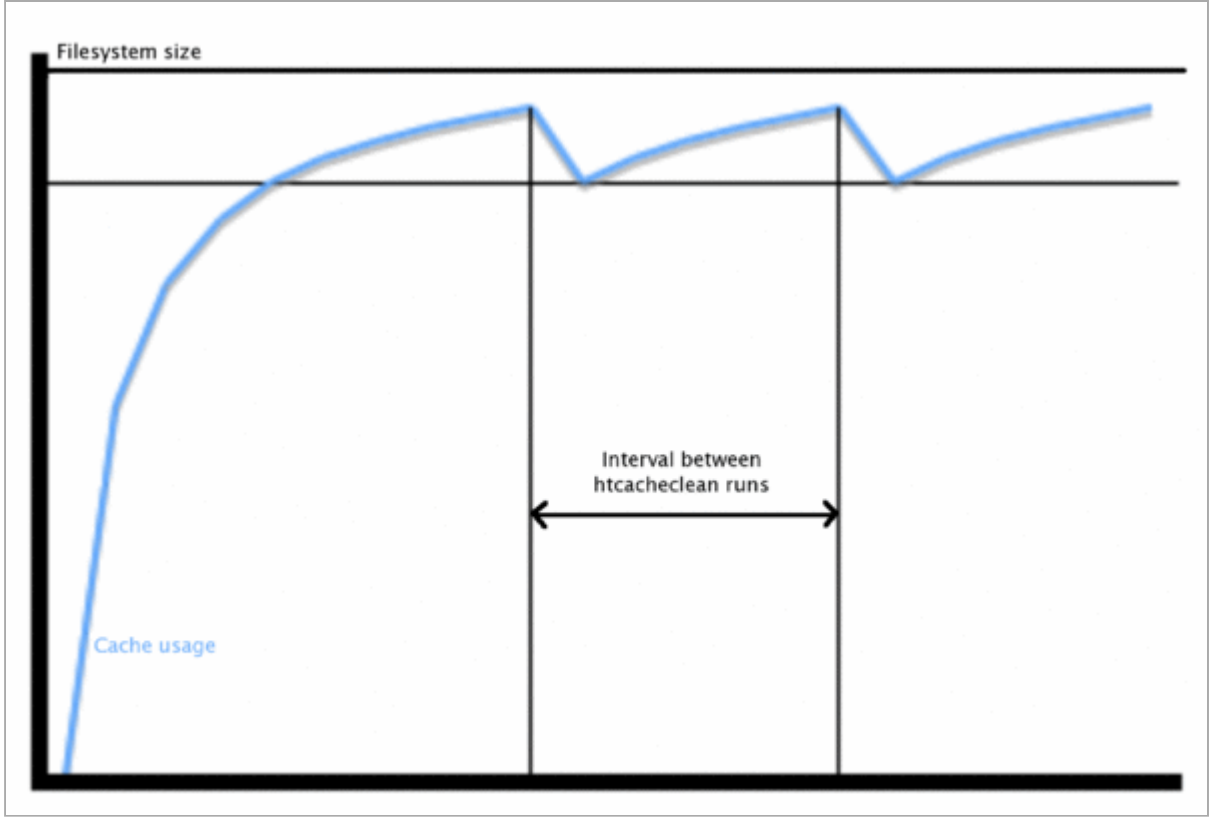


***Figure 1**: Typical cache growth / clean sequence.*

Because `mod_cache_disk` does not itself pay attention to how much space is used you should ensure that htcacheclean is configured to leave enough "grow room" following a clean.

Caching to memcached

Using the `mod_cache_socache` module, `mod_cache` can cache data from a variety of implementations (aka: "providers"). Using the `mod_socache_memcache` module, for example, one can specify that memcached is to be used as the the backend storage mechanism.

Typically the module will be configured as so:

```
CacheEnable socache /
CacheSocache memcache:memcd.example.com:11211
```

Additional `memcached` servers can be specified by appending them to the end of the `CacheSocache memcache:` line separated by commas:

```
CacheEnable socache /
CacheSocache memcache:mem1.example.com:11211,mem2.examp
```

This format is also used with the other various `mod_cache_socache` providers. For example:

```
CacheEnable socache /
CacheSocache shmcb:/path/to/datafile(512000)
```

```
CacheEnable socache /
CacheSocache dbm:/path/to/datafile
```

General Two-state Key/Value Shared Object Caching

Related Modules	Related Directives
`mod_authn_socache`	AuthnCacheSOCache
`mod_socache_dbm`	SSLSessionCache

mod_socache_dc	SSLStaplingCache
mod_socache_memcache	
mod_socache_shmcb	
mod_ssl	

The Apache HTTP server offers a low level shared object cache for caching information such as SSL sessions, or authentication credentials, within the socache interface.

Additional modules are provided for each implementation, offering the following backends:

mod_socache_dbm
: DBM based shared object cache.

mod_socache_dc
: Distcache based shared object cache.

mod_socache_memcache
: Memcache based shared object cache.

mod_socache_shmcb
: Shared memory based shared object cache.

Caching Authentication Credentials

Related Modules	Related Directives
mod_authn_socache	AuthnCacheSOCache

The mod_authn_socache module allows the result of authentication to be cached, relieving load on authentication backends.

Caching SSL Sessions

Related Modules	Related Directives
mod_ssl	SSLSessionCache
	SSLStaplingCache

The mod_ssl module uses the socache interface to provide a session cache and a stapling cache.

Specialized File Caching

Related Modules	Related Directives
mod_file_cache	CacheFile
	MMapFile

On platforms where a filesystem might be slow, or where file handles are expensive, the option exists to pre-load files into memory on startup.

On systems where opening files is slow, the option exists to open the file on startup and cache the file handle. These options can help on systems where access to static files is slow.

File-Handle Caching

The act of opening a file can itself be a source of delay, particularly on network filesystems. By maintaining a cache of open file descriptors for commonly served files, httpd can avoid this delay. Currently httpd provides one implementation of File-Handle Caching.

CacheFile

The most basic form of caching present in httpd is the file-handle caching provided by mod_file_cache. Rather than caching file-contents, this cache maintains a table of open file descriptors. Files to be cached in this manner are specified in the configuration file using the CacheFile directive.

The CacheFile directive instructs httpd to open the file when it is started and to re-use this file-handle for all subsequent access to this file.

```
CacheFile /usr/local/apache2/htdocs/index.html
```

If you intend to cache a large number of files in this manner, you must ensure that your operating system's limit for the number of open files is set appropriately.

Although using CacheFile does not cause the file-contents to be cached per-se, it does mean that if the file changes while httpd is running these changes will not be picked up. The file will be consistently served as it was when httpd was started.

If the file is removed while httpd is running, it will continue to maintain an open file descriptor and serve the file as it was when httpd was started. This usually also means that although the file will have been deleted, and not show up on the filesystem, extra free space will not be recovered until httpd is stopped and the file descriptor closed.

In-Memory Caching

Serving directly from system memory is universally the fastest method of serving content. Reading files from a disk controller or, even worse, from a remote network is orders of magnitude slower. Disk controllers usually involve physical processes, and network access is limited by your available bandwidth. Memory access on the other hand can take mere nano-seconds.

System memory isn't cheap though, byte for byte it's by far the most expensive type of storage and it's important to ensure that it is used efficiently. By caching files in memory you decrease the amount of memory available on the system. As we'll see, in the case of operating system caching, this is not so much of an issue, but when using httpd's own in-memory caching it is important to make sure that you do not allocate too much memory to a cache. Otherwise the system will be forced to swap out memory, which will likely degrade performance.

Operating System Caching

Almost all modern operating systems cache file-data in memory managed directly by the kernel. This is a powerful feature, and for the most part operating systems get it right. For example, on Linux, let's look at the difference in the time it takes to read a file for the first time and the second time;

```
colm@coroebus:~$ time cat testfile > /dev/null
real    0m0.065s
user    0m0.000s
sys     0m0.001s
colm@coroebus:~$ time cat testfile > /dev/null
real    0m0.003s
user    0m0.003s
sys     0m0.000s
```

Even for this small file, there is a huge difference in the amount of time it takes to read the file. This is because the kernel has cached the file contents in memory.

By ensuring there is "spare" memory on your system, you can ensure that more and more file-contents will be stored in this cache. This can be a very efficient means of in-memory caching, and involves no extra configuration of httpd at all.

Additionally, because the operating system knows when files are deleted or modified, it can automatically remove file contents from the cache when

necessary. This is a big advantage over httpd's in-memory caching which has no way of knowing when a file has changed.

Despite the performance and advantages of automatic operating system caching there are some circumstances in which in-memory caching may be better performed by httpd.

MMapFile Caching

mod_file_cache provides the MMapFile directive, which allows you to have httpd map a static file's contents into memory at start time (using the mmap system call). httpd will use the in-memory contents for all subsequent accesses to this file.

```
MMapFile /usr/local/apache2/htdocs/index.html
```

As with the CacheFile directive, any changes in these files will not be picked up by httpd after it has started.

The MMapFile directive does not keep track of how much memory it allocates, so you must ensure not to over-use the directive. Each httpd child process will replicate this memory, so it is critically important to ensure that the files mapped are not so large as to cause the system to swap memory.

Security Considerations

Authorization and Access Control

Using mod_cache in its default state where CacheQuickHandler is set to On is very much like having a caching reverse-proxy bolted to the front of the server. Requests will be served by the caching module unless it determines that the origin server should be queried just as an external cache would, and this drastically changes the security model of httpd.

As traversing a filesystem hierarchy to examine potential .htaccess files would be a very expensive operation, partially defeating the point of caching (to speed up requests), mod_cache makes no decision about whether a cached entity is authorised for serving. In other words; if mod_cache has cached some content, it will be served from the cache as long as that content has not expired.

If, for example, your configuration permits access to a resource by IP address you should ensure that this content is not cached. You can do this by using the CacheDisable directive, or mod_expires. Left unchecked, mod_cache - very much like a reverse proxy - would cache the content when served and then serve it to any client, on any IP address.

When the CacheQuickHandler directive is set to Off, the full set of request processing phases are executed and the security model remains unchanged.

Local exploits

As requests to end-users can be served from the cache, the cache itself can become a target for those wishing to deface or interfere with content. It is important to bear in mind that the cache must at all times be writable by the user which httpd is running as. This is in stark contrast to the usually recommended situation of maintaining all content unwritable by the Apache user.

If the Apache user is compromised, for example through a flaw in a CGI process, it is possible that the cache may be targeted. When using mod_cache_disk, it is relatively easy to insert or modify a cached entity.

This presents a somewhat elevated risk in comparison to the other types of attack it is possible to make as the Apache user. If you are using mod_cache_disk you should bear this in mind - ensure you upgrade httpd when security upgrades are

announced and run CGI processes as a non-Apache user using suEXEC if possible.

Cache Poisoning

When running httpd as a caching proxy server, there is also the potential for so-called cache poisoning. Cache Poisoning is a broad term for attacks in which an attacker causes the proxy server to retrieve incorrect (and usually undesirable) content from the origin server.

For example if the DNS servers used by your system running httpd are vulnerable to DNS cache poisoning, an attacker may be able to control where httpd connects to when requesting content from the origin server. Another example is so-called HTTP request-smuggling attacks.

This document is not the correct place for an in-depth discussion of HTTP request smuggling (instead, try your favourite search engine) however it is important to be aware that it is possible to make a series of requests, and to exploit a vulnerability on an origin webserver such that the attacker can entirely control the content retrieved by the proxy.

Denial of Service / Cachebusting

The Vary mechanism allows multiple variants of the same URL to be cached side by side. Depending on header values provided by the client, the cache will select the correct variant to return to the client. This mechanism can become a problem when an attempt is made to vary on a header that is known to contain a wide range of possible values under normal use, for example the `User-Agent` header. Depending on the popularity of the particular web site thousands or millions of duplicate cache entries could be created for the same URL, crowding out other entries in the cache.

In other cases, there may be a need to change the URL of a particular resource on every request, usually by adding a "cachebuster" string to the URL. If this content is declared cacheable by a server for a significant freshness lifetime, these entries can crowd out legitimate entries in a cache. While `mod_cache` provides a `CacheIgnoreURLSessionIdentifiers` directive, this directive should be used with care to ensure that downstream proxy or browser caches aren't subjected to the same denial of service issue.

Content Negotiation

Apache HTTPD supports content negotiation as described in the HTTP/1.1 specification. It can choose the best representation of a resource based on the browser-supplied preferences for media type, languages, character set and encoding. It also implements a couple of features to give more intelligent handling of requests from browsers that 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
```

httpd 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. httpd 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 HTTP Server 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 httpd

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 httpd 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

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

in the server configuration file.

Type map files should have the same name as the resource which they are describing, followed by the extension `.var`. In the examples shown below, the resource is named `foo`, so the type map file is named `foo.var`.

This file should 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.

URIs in this file are relative to the location of the type map file. Usually, these files will be located in the same directory as the type map file, but this is not required. You may provide absolute or relative URIs for any file located on the same server as the map file.

```
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 available in the mod_negotiation typemap documentation.

Multiviews

`MultiViews` is a per-directory option, meaning it can be set with an `Options` directive within a `<Directory>`, `<Location>` or `<Files>` section in `httpd.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 directory does not have an extension recognized by `mod_mime` to designate its Charset, Content-Type, Language, or Encoding, then the result depends on the setting of the `MultiViewsMatch` directive. This directive determines whether handlers, filters, and other extension types can participate in MultiViews negotiation.

The Negotiation Methods

After httpd 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 httpd'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 httpd algorithm** is used in the normal case. The httpd algorithm is explained in more detail below. When this algorithm is used, httpd can sometimes 'fiddle' the quality factor of a particular dimension to achieve a better result. The ways httpd 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 httpd 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.

httpd Negotiation Algorithm

httpd 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 variants 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

httpd sometimes changes the quality values from what would be expected by a strict interpretation of the httpd 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. 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. Using explicit quality values, what the browser really wants is something like:

```
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, httpd 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 explicit information to start with work as expected.

Language Negotiation Exceptions

New in httpd 2.0, some exceptions have been added to the negotiation algorithm to allow graceful fallback when language negotiation fails to find a match.

When a client requests a page on your server, but the server cannot find a single page that matches the `Accept-language` sent by the browser, the server will return either a "No Acceptable Variant" or "Multiple Choices" response to the client. To avoid these error messages, it is possible to configure httpd to ignore the `Accept-language` in these cases and provide a document that does not explicitly match the client's request. The `ForceLanguagePriority` directive

can be used to override one or both of these error messages and substitute the servers judgement in the form of the `LanguagePriority` directive.

The server will also attempt to match language-subsets when no other match can be found. For example, if a client requests documents with the language `en-GB` for British English, the server is not normally allowed by the HTTP/1.1 standard to match that against a document that is marked as simply `en`. (Note that it is almost surely a configuration error to include `en-GB` and not `en` in the `Accept-Language` header, since it is very unlikely that a reader understands British English, but doesn't understand English in general. Unfortunately, many current clients have default configurations that resemble this.) However, if no other language match is possible and the server is about to return a "No Acceptable Variants" error or fallback to the `LanguagePriority`, the server will ignore the subset specification and match `en-GB` against `en` documents. Implicitly, httpd will add the parent language to the client's acceptable language list with a very low quality value. But note that if the client requests "en-GB; q=0.9, fr; q=0.8", and the server has documents designated "en" and "fr", then the "fr" document will be returned. This is necessary to maintain compliance with the HTTP/1.1 specification and to work effectively with properly configured clients.

In order to support advanced techniques (such as cookies or special URL-paths) to determine the user's preferred language, since httpd 2.0.47 `mod_negotiation` recognizes the environment variable `prefer-language`. If it exists and contains an appropriate language tag, `mod_negotiation` will try to select a matching variant. If there's no such variant, the normal negotiation process applies.

Example

```
SetEnvIf Cookie "language=(.+)" prefer-language=$1
Header append Vary cookie
```

Extensions to Transparent Content Negotiation

httpd 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 a 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, httpd normally marks all responses that are returned after content negotiation as non-cacheable by HTTP/1.0 clients. httpd 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.

For HTTP/1.1 clients, httpd sends a `Vary` HTTP response header to indicate the negotiation dimensions for the response. Caches can use this information to determine whether a subsequent request can be served from the local copy. To encourage a cache to use the local copy regardless of the negotiation dimensions, set the `force-no-vary` environment variable.

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. Modules will 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`).

Alternatively, the modules can be statically compiled into the `httpd` binary when the server is built.

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

Implementation

Related Modules	Related Directives
mod_so	LoadModule

The DSO support for loading individual Apache httpd modules is based on a module named `mod_so` which must be statically compiled into the Apache httpd core. It is the only module besides `core` which cannot be put into a DSO itself. Practically all other distributed Apache httpd modules will then be placed into a DSO. After a module is compiled into a DSO named `mod_foo.so` you can use `mod_so`'s `LoadModule` directive in your `httpd.conf` file to load this module at server startup or restart.

The DSO builds for individual modules can be disabled via `configure`'s `--enable-mods-static` option as discussed in the install documentation.

To simplify this creation of DSO files for Apache httpd modules (especially for third-party modules) a support program named `apxs` (*APache eXtenSion*) is available. It can be used to build DSO based modules *outside of* the Apache httpd source tree. The idea is simple: When installing Apache HTTP Server the `configure`'s `make install` procedure installs the Apache httpd 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 httpd module sources without the Apache httpd 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 HTTP Server 2.x, here is a short and concise summary:

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

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

2. Configure Apache HTTP Server with all modules enabled. Only a basic set will be loaded during server startup. You can change the set of loaded modules by activating or deactivating the `LoadModule` directives in `httpd.conf`.

```
$ ./configure --enable-mods-shared=all
$ make install
```

3. Some modules are only useful for developers and will not be build. when using the module set *all*. To build all available modules including developer modules use *reallyall*. In addition the `LoadModule` directives for all built modules can be activated via the configure option `--enable-load-all-modules`.

```
$ ./configure --enable-mods-shared=reallyall --
enable-load-all-modules
$ make install
```

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

```
$ cd /path/to/3rdparty
$ apxs -cia mod_foo.c
```

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

See the apxs documentation for more details.

Background

On modern Unix derivatives there exists a mechanism 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.

Advantages and Disadvantages

The above DSO based features have the following advantages:

- The server package is more flexible at run-time because the server process can be assembled at run-time via `LoadModule` `httpd.conf` configuration directives instead of `configure` options at build-time. For instance, this way one is able to run different server instances (standard & SSL version, minimalistic & dynamic version [mod_perl, mod_php], *etc.*) with only one Apache httpd installation.
- The server package can be easily extended with third-party modules even after installation. This is a great benefit for vendor package maintainers, who can create an Apache httpd core package and additional packages containing extensions like PHP, mod_perl, mod_security, *etc.*
- Easier Apache httpd module prototyping, because with the DSO/`apxs` pair you can both work outside the Apache httpd 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 HTTP Server.

DSO has the following disadvantages:

- 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 httpd core, from the C library (`libc`) and all other dynamic or static libraries used by the Apache httpd 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 httpd core itself already contains a reference to it or loading the code yourself via `dlopen()`.

Environment Variables in Apache

There are two kinds of environment variables that affect the Apache HTTP Server.

First, there are the environment variables controlled by the underlying operating system. These are set before the server starts. They can be used in expansions in configuration files, and can optionally be passed to CGI scripts and SSI using the PassEnv directive.

Second, the Apache HTTP Server provides a mechanism for storing information in named variables that are also 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_cache	BrowserMatch
mod_env	BrowserMatchNoCase
mod_rewrite	PassEnv
mod_setenvif	RewriteRule
mod_unique_id	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.
- A special case are HTTP headers which are passed to CGI scripts and the like via environment variables (see below). They are converted to uppercase and only dashes are replaced with underscores; if the header contains any other (invalid) character, the whole header is silently dropped. See below for a workaround.
- The `SetEnv` directive runs late during request processing meaning that directives such as `SetEnvIf` and `RewriteCond` will not see the variables set with it.
- When the server looks up a path via an internal subrequest such as looking for a `DirectoryIndex` or generating a directory listing with `mod_autoindex`, per-request environment variables are *not* inherited in the subrequest. Additionally, `SetEnvIf` directives are not separately evaluated in the subrequest due to the API phases `mod_setenvif` takes action in.

Using Environment Variables

Related Modules	Related Directives
mod_authz_host	Require
mod_cgi	CustomLog
mod_ext_filter	Deny
mod_headers	ExtFilterDefine
mod_include	Header
mod_log_config	LogFormat
mod_rewrite	RewriteCond
	RewriteRule

CGI Scripts

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

SSI Pages

Server-parsed (SSI) documents processed by `mod_include`'s `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.

External Filter Activation

External filters configured by `mod_ext_filter` using the `ExtFilterDefine` directive can by activated conditional on an environment variable using the `disableenv=` and `enableenv=` options.

URL Rewriting

The `%{ENV:variable}` 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-gzip

If you have the `DEFLATE` filter activated, this environment variable will ignore the accept-encoding setting of your browser and will send compressed output unconditionally.

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

gzip-only-text/html

When set to a value of "1", this variable disables the `DEFLATE` output filter provided by `mod_deflate` for content-types other than `text/html`. If you'd rather use statically compressed files, `mod_negotiation` evaluates the variable as well (not only for gzip, but for all encodings that differ from "identity").

no-gzip

When set, the `DEFLATE` filter of `mod_deflate` will be turned off and `mod_negotiation` will refuse to deliver encoded resources.

no-cache

Available in versions 2.2.12 and later

When set, `mod_cache` will not save an otherwise cacheable response. This environment variable does not influence whether a response already in the cache will be served for the current request.

nokeepalive

This disables `KeepAlive` when set.

prefer-language

This influences `mod_negotiation`'s behaviour. If it contains a language tag (such as `en`, `ja` or `x-klingon`), `mod_negotiation` tries to deliver a variant with that language. If there's no such variant, the normal negotiation process applies.

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.

suppress-error-charset

Available in versions after 2.0.54

When Apache issues a redirect in response to a client request, the response includes some actual text to be displayed in case the client can't (or doesn't) automatically follow the redirection. Apache ordinarily labels this text according to the character set which it uses, which is ISO-8859-1.

However, if the redirection is to a page that uses a different character set, some broken browser versions will try to use the character set from the redirection text rather than the actual page. This can result in Greek, for instance, being incorrectly rendered.

Setting this environment variable causes Apache to omit the character set for the redirection text, and these broken browsers will then correctly use that of the destination page.

> **Security note**
>
> Sending error pages without a specified character set may allow a cross-site-scripting attack for existing browsers (MSIE) which do not

follow the HTTP/1.1 specification and attempt to "guess" the character set from the content. Such browsers can be easily fooled into using the UTF-7 character set, and UTF-7 content from input data (such as the request-URI) will not be escaped by the usual escaping mechanisms designed to prevent cross-site-scripting attacks.

force-proxy-request-1.0, proxy-nokeepalive, proxy-sendchunked, proxy-sendcl, proxy-chain-auth, proxy-interim-response, proxy-initial-not-pooled

These directives alter the protocol behavior of mod_proxy. See the mod_proxy and mod_proxy_http documentation for more details.

Examples

Passing broken headers to CGI scripts

Starting with version 2.4, Apache is more strict about how HTTP headers are converted to environment variables in mod_cgi and other modules: Previously any invalid characters in header names were simply translated to underscores. This allowed for some potential cross-site-scripting attacks via header injection (see Unusual Web Bugs, slide 19/20).

If you have to support a client which sends broken headers and which can't be fixed, a simple workaround involving mod_setenvif and mod_headers allows you to still accept these headers:

```
#
# The following works around a client sending a broken
# header.
#
SetEnvIfNoCase ^Accept.Encoding$ ^(.*)$ fix_accept_enc
RequestHeader set Accept-Encoding %{fix_accept_encodin
```

Changing protocol behavior with misbehaving clients

Earlier versions recommended that the following lines be included in httpd.conf to deal with known client problems. Since the affected clients are no longer seen in the wild, this configuration is likely no-longer necessary.

```
#
# The following directives modify normal HTTP response
# The first directive disables keepalive for Netscape 2
# spoof it. There are known problems with these browser
# The second directive is for Microsoft Internet Explor
# which has a broken HTTP/1.1 implementation and does r
# support keepalive when it is used on 301 or 302 (redi
#
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\.0b2;" nokeepalive downgrade-1.0 f

#
# The following directive disables HTTP/1.1 responses t
# are in violation of the HTTP/1.0 spec by not being ab
# 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" common 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_ref
# Allow browsers that do not send Referer info
SetEnvIf Referer "^$" local_referal
<Directory "/web/images">
    Require env local_referal
</Directory>
```

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

Log Files

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

Overview

Related Modules	Related Directives
mod_log_config	
mod_log_forensic	
mod_logio	
mod_cgi	

The Apache HTTP Server provides a variety of different mechanisms for logging everything that happens on your server, from the initial request, through the URL mapping process, to the final resolution of the connection, including any errors that may have occurred in the process. In addition to this, third-party modules may provide logging capabilities, or inject entries into the existing log files, and applications such as CGI programs, or PHP scripts, or other handlers, may send messages to the server error log.

In this document we discuss the logging modules that are a standard part of the http server.

Security Warning

Anyone who can write to the directory where Apache httpd 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 Modules	Related Directives
core	ErrorLog
	ErrorLogFormat
	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 defined by the ErrorLogFormat directive, with which you can customize what values are logged. A default is format defined if you don't specify one. A typical log message follows:

```
[Fri Sep 09 10:42:29.902022 2011] [core:error] [pid
35708:tid 4328636416] [client 72.15.99.187] File does not
exist: /usr/local/apache2/htdocs/favicon.ico
```

The first item in the log entry is the date and time of the message. The next is the module producing the message (core, in this case) and the severity level of that message. This is followed by the process ID and, if appropriate, the thread ID, of the process that experienced the condition. Next, we have the client address that made the request. And finally is the detailed error message, which in this case indicates a request for a file that did not exist.

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.

Putting a `%L` token in both the error log and the access log will produce a log entry ID with which you can correlate the entry in the error log with the entry in the access log. If mod_unique_id is loaded, its unique request ID will be used as the log entry ID, too.

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
```

Per-module logging

The LogLevel directive allows you to specify a log severity level on a per-module basis. In this way, if you are troubleshooting a problem with just one particular module, you can turn up its logging volume without also getting the details of other modules that you're not interested in. This is particularly useful for modules such as mod_proxy or mod_rewrite where you want to know details about what it's trying to do.

Do this by specifying the name of the module in your `LogLevel` directive:

```
LogLevel info rewrite:trace5
```

This sets the main `LogLevel` to info, but turns it up to `trace5` for mod_rewrite.

> This replaces the per-module logging directives, such as `RewriteLog`, that were present in earlier versions of the server.

Access Log

Related Modules	Related Directives
mod_log_config	CustomLog
mod_setenvif	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.

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` format strings.

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 backslash 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 part will be "`-`" just like the previous one.

`[10/Oct/2000:13:55:36 -0700]` (`%t`)
: The time that the request was received. 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 either as in `strftime(3)` from the C standard library, or one of the supported special tokens. For details see the `mod_log_config` format strings.

`"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 part 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\" \'
CustomLog "log/access_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 Logs

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
```

In a caching scenario one would want to know about the efficiency of the cache. A very simple method to find this out would be:

```
SetEnv CACHE_MISS 1
LogFormat "%h %l %u %t "%r " %>s %b %{CACHE_MISS}e" com
CustomLog "logs/access_log" common-cache
```

mod_cache will run before mod_env and, when successful, will deliver the content without it. In that case a cache hit will log -, while a cache miss will log 1.

In addition to the env= syntax, LogFormat supports logging values conditional upon the HTTP response code:

```
LogFormat "%400,501{User-agent}i" browserlog
LogFormat "%!200,304,302{Referer}i" refererlog
```

In the first example, the User-agent will be logged if the HTTP status code is 400 or 501. In other cases, a literal "-" will be logged instead. Likewise, in the second example, the Referer will be logged if the HTTP status code is **not** 200, 204, or 302. (Note the "!" before the status codes.

Although we have just shown that conditional logging is very powerful and flexible, 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 httpd 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. The server 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.

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/a
```

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.

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.

By default the piped log process is spawned without invoking a shell. Use "|$" instead of "|" to spawn using a shell (usually with /bin/sh -c):

```
# Invoke "rotatelogs" using a shell
CustomLog "|$/usr/local/apache/bin/rotatelogs   /var/lo
```

This was the default behaviour for Apache 2.2. Depending on the shell specifics this might lead to an additional shell process for the lifetime of the logging pipe program and signal handling problems during restart. For compatibility reasons with Apache 2.2 the notation "||" is also supported and equivalent to using "|".

Windows note

Note that on Windows, you may run into problems when running many piped logger processes, especially when HTTPD is running as a service. This is caused by running out of desktop heap space. The desktop heap space given to each service is specified by the third argument to the SharedSection parameter in the HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\SessionManager\SubSystems\Windows registry value. **Change this value with care**; the normal caveats for changing the Windows registry apply, but you might also exhaust the desktop heap pool if the number is adjusted too high.

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.

Other Log Files

Related Modules	Related Directives
mod_logio	LogFormat
mod_log_config	BufferedLogs
mod_log_forensic	ForensicLog
mod_cgi	PidFile
	ScriptLog
	ScriptLogBuffer
	ScriptLogLength

Logging actual bytes sent and received

mod_logio adds in two additional LogFormat fields (%I and %O) that log the actual number of bytes received and sent on the network.

Forensic Logging

mod_log_forensic provides for forensic logging of client requests. Logging is done before and after processing a request, so the forensic log contains two log lines for each request. The forensic logger is very strict with no customizations. It can be an invaluable debugging and security tool.

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.

Mapping URLs to Filesystem Locations

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

Related Modules and Directives

Related Modules	Related Directives
mod_actions	Alias
mod_alias	AliasMatch
mod_autoindex	CheckSpelling
mod_dir	DirectoryIndex
mod_imagemap	DocumentRoot
mod_negotiation	ErrorDocument
mod_proxy	Options
mod_rewrite	ProxyPass
mod_speling	ProxyPassReverse
mod_userdir	ProxyPassReverseCookieDomain
mod_vhost_alias	ProxyPassReverseCookiePath
	Redirect
	RedirectMatch
	RewriteCond
	RewriteRule
	ScriptAlias
	ScriptAliasMatch
	UserDir

DocumentRoot

In deciding what file to serve for a given request, httpd'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.

For example, if DocumentRoot were set to `/var/www/html` then a request for `http://www.example.com/fish/guppies.html` would result in the file `/var/www/html/fish/guppies.html` being served to the requesting client.

If a directory is requested (i.e. a path ending with `/`), the file served from that directory is defined by the DirectoryIndex directive. For example, if DocumentRoot were set as above, and you were to set:

```
DirectoryIndex index.html index.php
```

Then a request for `http://www.example.com/fish/` will cause httpd to attempt to serve the file `/var/www/html/fish/index.html`. In the event that that file does not exist, it will next attempt to serve the file `/var/www/html/fish/index.php`.

If neither of these files existed, the next step is to attempt to provide a directory index, if mod_autoindex is loaded and configured to permit that.

httpd 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.

The DocumentRoot directive is set in your main server configuration file (httpd.conf) and, possibly, once per additional Virtual Host you create.

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. httpd 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, httpd 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 "^/~([a-zA-Z0-9]+)/cgi-bin/(.+)"    "/h
```

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/([a-zA-Z0-9]+)(/(.*))?$"    "/home/
```

URL Redirection

The configuration directives discussed in the above sections tell httpd 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/ba
```

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.

httpd 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.co
```

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

```
RedirectMatch temp ".*"  "http://othersite.example.com/
```

Reverse Proxy

httpd also allows you to bring remote documents into the URL space of the local server. This technique is called *reverse proxying* because the web server acts like a proxy server by fetching the documents from a remote server and returning them to the client. It is different from normal (forward) proxying because, to the client, it appears the documents originate at the reverse proxy server.

In the following example, when clients request documents under the /foo/ directory, the server fetches those documents from the /bar/ directory on internal.example.com and returns them to the client as if they were from the local server.

```
ProxyPass         "/foo/" "http://internal.example.com/b
ProxyPassReverse  "/foo/" "http://internal.example.com/b
ProxyPassReverseCookieDomain internal.example.com publi
ProxyPassReverseCookiePath "/foo/" "/bar/"
```

The ProxyPass configures the server to fetch the appropriate documents, while the ProxyPassReverse directive rewrites redirects originating at internal.example.com so that they target the appropriate directory on the local server. Similarly, the ProxyPassReverseCookieDomain and ProxyPassReverseCookiePath rewrite cookies set by the backend server.

It is important to note, however, that links inside the documents will not be rewritten. So any absolute links on internal.example.com will result in the client breaking out of the proxy server and requesting directly from

internal.example.com. You can modify these links (and other content) in a page as it is being served to the client using mod_substitute.

```
Substitute s/internal\.example\.com/www.example.com/i
```

For more sophisticated rewriting of links in HTML and XHTML, the mod_proxy_html module is also available. It allows you to create maps of URLs that need to be rewritten, so that complex proxying scenarios can be handled.

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 can 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. The rewriting engine is capable of performing all three types of mappings discussed above: internal redirects (aliases), external redirects, and proxying. Many practical examples employing mod_rewrite are discussed in the detailed mod_rewrite documentation.

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. httpd 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.

mod_dir provides FallbackResource, which can be used to map virtual URIs to a real resource, which then serves them. This is a very useful replacement for mod_rewrite when implementing a 'front controller'

If all attempts to locate the content fail, httpd 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 document.

Other URL Mapping Modules

Other modules available for URL mapping include:

- mod_actions - Maps a request to a CGI script based on the request method, or resource MIME type.
- mod_dir - Provides basic mapping of a trailing slash into an index file such as index.html.

- mod_imagemap - Maps a request to a URL based on where a user clicks on an image embedded in a HTML document.
- mod_negotiation - Selects an appropriate document based on client preferences such as language or content compression.

Apache Performance Tuning

Warning

This document is partially out of date and might be inaccurate.

Apache 2.4 is a general-purpose webserver, designed to provide a balance of flexibility, portability, and performance. Although it has not been designed specifically to set benchmark records, Apache 2.4 is capable of high performance in many real-world situations.

This document describes the options that a server administrator can configure to tune the performance of an Apache 2.4 installation. Some of these configuration options enable the httpd to better take advantage of the capabilities of the hardware and OS, while others allow the administrator to trade functionality for speed.

Hardware and Operating System Issues

The single biggest hardware issue affecting webserver performance is RAM. A webserver should never ever have to swap, as 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 `MaxRequestWorkers` setting so that your server does not spawn so many children that it starts swapping. The procedure for doing this is simple: determine the size of your average Apache process, by looking at your process list via a tool such as `top`, and divide this into your total available memory, leaving some room for other processes.

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 some guidelines that have proven generally useful are:

- Run the latest stable release and patch level of the operating system that you choose. Many OS suppliers have introduced significant performance improvements to their TCP stacks and thread libraries in recent years.
- If your OS supports a `sendfile(2)` system call, make sure you install the release and/or patches needed to enable it. (With Linux, for example, this means using Linux 2.4 or later. For early releases of Solaris 8, you may need to apply a patch.) On systems where it is available, `sendfile` enables Apache to deliver static content faster and with lower CPU utilization.

Run-Time Configuration Issues

Related Modules	Related Directives
mod_dir	AllowOverride
mpm_common	DirectoryIndex
mod_status	HostnameLookups
	EnableMMAP
	EnableSendfile
	KeepAliveTimeout
	MaxSpareServers
	MinSpareServers
	Options
	StartServers

HostnameLookups and other DNS considerations

Prior to Apache 1.3, `HostnameLookups` defaulted to `On`. causing an extra latency penalty for every request due to a DNS lookup to complete before the request was finished. In Apache 2.4 this setting defaults to `Off`. If you need to have addresses in your log files resolved to hostnames, please consider post-processing rather than forcing Apache to do it in the first place. It is recommended that you do this sort of post-processing of your log files on some machine other than the production web server machine, in order that this activity not adversely affect server performance.

If you use any `Allow from domain` or `Deny from domain` directives (i.e., using a hostname, or a domain name, rather than an IP address) then you will pay for two DNS lookups (a reverse, followed by a forward lookup to make sure that the reverse is not being spoofed). For best performance, whenever it is possible, use IP addresses rather than domain names.

> **Warning:**
> Please use the `Require` directive with Apache 2.4; more info in the related upgrading guide.

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:

```
<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.

FollowSymLinks and SymLinksIfOwnerMatch

Wherever in your URL-space you do not have an `Options FollowSymLinks`, or you do have an `Options SymLinksIfOwnerMatch`, Apache will need 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.htm
```

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.

If your site needs content negotiation, consider using `type-map` files, rather than the `Options MultiViews` directive to accomplish the negotiation. See the Content Negotiation documentation for a full discussion of the methods of negotiation, and instructions for creating `type-map` files.

Memory-mapping

In situations where Apache 2.x needs to look at the contents of a file being delivered--for example, when doing server-side-include processing--it normally memory-maps the file if the OS supports some form of `mmap(2)`.

On some platforms, this memory-mapping improves performance. However, there are cases where memory-mapping can hurt the performance or even the stability of the httpd:

- On some operating systems, `mmap` does not scale as well as `read(2)` when the number of CPUs increases. On multiprocessor Solaris servers, for example, Apache 2.x sometimes delivers server-parsed files faster when `mmap` is disabled.
- If you memory-map a file located on an NFS-mounted filesystem and a process on another NFS client machine deletes or truncates the file, your process may get a bus error the next time it tries to access the mapped file content.

For installations where either of these factors applies, you should use `EnableMMAP off` to disable the memory-mapping of delivered files. (Note: This directive can be overridden on a per-directory basis.)

Sendfile

In situations where Apache 2.x can ignore the contents of the file to be delivered -- for example, when serving static file content -- it normally uses the kernel sendfile support for the file if the OS supports the `sendfile(2)` operation.

On most platforms, using sendfile improves performance by eliminating separate read and send mechanics. However, there are cases where using sendfile can harm the stability of the httpd:

- Some platforms may have broken sendfile support that the build system did not detect, especially if the binaries were built on another box and moved to such a machine with broken sendfile support.
- With an NFS-mounted filesystem, the kernel may be unable to reliably serve the network file through its own cache.

For installations where either of these factors applies, you should use `EnableSendfile off` to disable sendfile delivery of file contents. (Note: This directive can be overridden on a per-directory basis.)

Recycle child processes

`MaxConnectionsPerChild` limits the numbers of connections that a child process can handle during its lifetime (by default set to `0` - unlimited). This affects all the MPMs, even the ones using threads. For example, each process created by the `worker` MPM spawns multiple threads that will handle connections, but this does not influence the overall count. It only means that the sum of requests handled by all the threads spawned by a single process will be counted against the `MaxConnectionsPerChild` value.

`MaxConnectionsPerChild` should not have any limit in the optimal use case, since there should not be any reason to force a process kill other than software bugs causing memory leaks or excessive CPU usage.

When keep-alives are in use, a process (or a thread spawned by a process) will be kept busy doing nothing but waiting for more requests on the already open connection. The default `KeepAliveTimeout` of `5` seconds attempts to minimize this effect. The tradeoff here is between network bandwidth and server resources.

Compile-Time Configuration Issues

Choosing an MPM

Apache 2.x supports pluggable concurrency models, called Multi-Processing Modules (MPMs). When building Apache, you must choose an MPM to use. There are platform-specific MPMs for some platforms: `mpm_netware`, `mpmt_os2`, and `mpm_winnt`. For general Unix-type systems, there are several MPMs from which to choose. The choice of MPM can affect the speed and scalability of the httpd:

- The `worker` MPM uses multiple child processes with many threads each. Each thread handles one connection at a time. Worker generally is a good choice for high-traffic servers because it has a smaller memory footprint than the prefork MPM.
- The `event` MPM is threaded like the Worker MPM, but is designed to allow more requests to be served simultaneously by passing off some processing work to supporting threads, freeing up the main threads to work on new requests.

- The `prefork` MPM uses multiple child processes with one thread each. Each process handles one connection at a time. On many systems, prefork is comparable in speed to worker, but it uses more memory. Prefork's threadless design has advantages over worker in some situations: it can be used with non-thread-safe third-party modules, and it is easier to debug on platforms with poor thread debugging support.

For more information on these and other MPMs, please see the MPM documentation.

Modules

Since memory usage is such an important consideration in performance, you should attempt to eliminate modules that you are not actually using. If you have built the modules as DSOs, eliminating modules is a simple matter of commenting out the associated `LoadModule` directive for that module. This allows you to experiment with removing modules and seeing if your site still functions in their absence.

If, on the other hand, you have modules statically linked into your Apache binary, you will need to recompile Apache in order to remove unwanted modules.

An associated question that arises here is, of course, what modules you need, and which ones you don't. The answer here will, of course, vary from one web site to another. However, the *minimal* list of modules which you can get by with tends to include `mod_mime`, `mod_dir`, and `mod_log_config`. `mod_log_config` is, of course, optional, as you can run a web site without log files. This is, however, not recommended.

Atomic Operations

Some modules, such as `mod_cache` and recent development builds of the worker MPM, use APR's atomic API. This API provides atomic operations that can be used for lightweight thread synchronization.

By default, APR implements these operations using the most efficient mechanism available on each target OS/CPU platform. Many modern CPUs, for example, have an instruction that does an atomic compare-and-swap (CAS) operation in hardware. On some platforms, however, APR defaults to a slower, mutex-based implementation of the atomic API in order to ensure compatibility with older CPU models that lack such instructions. If you are building Apache for one of these platforms, and you plan to run only on newer CPUs, you can select a faster atomic implementation at build time by configuring Apache with the `--enable-nonportable-atomics` option:

```
./buildconf
./configure --with-mpm=worker --enable-nonportable-
atomics=yes
```

The `--enable-nonportable-atomics` option is relevant for the following platforms:

- Solaris on SPARC
  By default, APR uses mutex-based atomics on Solaris/SPARC. If you configure with `--enable-nonportable-atomics`, however, APR generates code that uses a SPARC v8plus opcode for fast hardware compare-and-swap. If you configure Apache with this option, the atomic operations will be more efficient (allowing for lower CPU utilization and higher concurrency), but the resulting executable will run only on UltraSPARC chips.
- Linux on x86
  By default, APR uses mutex-based atomics on Linux. If you configure with `--enable-nonportable-atomics`, however, APR generates code that

uses a 486 opcode for fast hardware compare-and-swap. This will result in more efficient atomic operations, but the resulting executable will run only on 486 and later chips (and not on 386).

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

> **Warning:**
>
> This section has not been fully updated to take into account changes made in the 2.x version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

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; +
              FD_SET (i, &accept_fds);
            }
            rc = select (last_socket+1, &accept_fds, N
            if (rc < 1) continue;
            new_connection = -1;
            for (i = first_socket; i <= last_socket; +
              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; +
              FD_SET (i, &accept_fds);
            }
            rc = select (last_socket+1, &accept_fds, N
            if (rc < 1) continue;
            new_connection = -1;
            for (i = first_socket; i <= last_socket; +
              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_connect
        }
```

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.

The `Mutex` directive can be used to change the mutex implementation of the `mpm-accept` mutex at run-time. Special considerations for different mutex implementations are documented with that directive.

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

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

accept Serialization - Single Socket

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

they discover there's no connection for them. This spinning is hidden from the user-land code, but it's there nonetheless. This can result in the same load-spiking wasteful behavior that a non-blocking solution to the multiple sockets case can.

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

Lingering Close

As discussed in draft-ietf-http-connection-00.txt section 8, in order for an HTTP server to **reliably** implement the protocol, it needs to shut down each direction of the communication independently. (Recall that a TCP connection is bi-directional. Each half is independent of the other.)

When this feature was added to Apache, it caused a flurry of problems on various versions of Unix because of 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 to accomplish 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_b
                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 behavior of earlier versions of Apache on Linux.

DYNAMIC_MODULE_LIMIT

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

Appendix: Detailed Analysis of a Trace

Here is a system call trace of Apache 2.0.38 with the worker MPM on Solaris 8. This trace was collected using:

```
truss -l -p httpd_child_pid.
```

The `-l` option tells truss to log the ID of the LWP (lightweight process--Solaris' form of kernel-level thread) that invokes each system call.

Other systems may have different system call tracing utilities such as `strace`, `ktrace`, or `par`. They all produce similar output.

In this trace, a client has requested a 10KB static file from the httpd. Traces of non-static requests or requests with content negotiation look wildly different (and quite ugly in some cases).

```
/67:    accept(3, 0x00200BEC, 0x00200C0C, 1) (sleeping...)
/67:    accept(3, 0x00200BEC, 0x00200C0C, 1)            = 9
```

In this trace, the listener thread is running within LWP #67.

> Note the lack of `accept(2)` serialization. On this particular platform, the worker MPM uses an unserialized accept by default unless it is listening on multiple ports.

```
/65:    lwp_park(0x00000000, 0)                         = 0
/67:    lwp_unpark(65, 1)                               = 0
```

Upon accepting the connection, the listener thread wakes up a worker thread to do the request processing. In this trace, the worker thread that handles the request is mapped to LWP #65.

```
/65:    getsockname(9, 0x00200BA4, 0x00200BC4, 1)       = 0
```

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

```
/65:    brk(0x002170E8)                                 = 0
/65:    brk(0x002190E8)                                 = 0
```

The `brk(2)` calls allocate memory from the heap. It is rare to see these in a system call trace, because the httpd uses custom memory allocators (`apr_pool` and `apr_bucket_alloc`) for most request processing. In this trace, the httpd has just been started, so it must call `malloc(3)` to get the blocks of raw memory with which to create the custom memory allocators.

```
/65:    fcntl(9, F_GETFL, 0x00000000)                   = 2
/65:    fstat64(9, 0xFAF7B818)                          = 0
/65:    getsockopt(9, 65535, 8192, 0xFAF7B918, 0xFAF7B910, 2190656) = 0
/65:    fstat64(9, 0xFAF7B818)                          = 0
/65:    getsockopt(9, 65535, 8192, 0xFAF7B918, 0xFAF7B914, 2190656) = 0
/65:    setsockopt(9, 65535, 8192, 0xFAF7B918, 4, 2190656) = 0
/65:    fcntl(9, F_SETFL, 0x00000082)                   = 0
```

Next, the worker thread puts the connection to the client (file descriptor 9) in non-blocking mode. The `setsockopt(2)` and `getsockopt(2)` calls are a side-effect of how Solaris' libc handles `fcntl(2)` on sockets.

```
/65:    read(9, " G E T   / 1 0 k . h t m"..., 8000)    = 97
```

The worker thread reads the request from the client.

```
/65:    stat("/var/httpd/apache/httpd-8999/htdocs/10k.html", 0xFAF7B978) = 0
/65:    open("/var/httpd/apache/httpd-8999/htdocs/10k.html", O_RDONLY) = 10
```

This httpd has been configured with `Options FollowSymLinks` and `AllowOverride None`. Thus it doesn't need to `lstat(2)` each directory in the path leading up to the requested file, nor check for `.htaccess` files. It simply calls `stat(2)` to verify that the file: 1) exists, and 2) is a regular file, not a directory.

```
/65:    sendfilev(0, 9, 0x00200F90, 2, 0xFAF7B53C)      = 10269
```

In this example, the httpd is able to send the HTTP response header and the requested file with a single `sendfilev(2)` system call. Sendfile semantics vary among operating systems. On some other systems, it is necessary to do a `write(2)` or `writev(2)` call to send the headers before calling `sendfile(2)`.

```
/65:    write(4, " 1 2 7 . 0 . 0 . 1   -  "..., 78)      = 78
```

This `write(2)` call records the request in the access log. Note that one thing missing from this trace is a `time(2)` call. Unlike Apache 1.3, Apache 2.x uses `gettimeofday(3)` to look up the time. On some operating systems, like Linux

or Solaris, `gettimeofday` has an optimized implementation that doesn't require as much overhead as a typical system call.

```
/65:    shutdown(9, 1, 1)                               = 0
/65:    poll(0xFAF7B980, 1, 2000)                       = 1
/65:    read(9, 0xFAF7BC20, 512)                        = 0
/65:    close(9)                                        = 0
```

The worker thread does a lingering close of the connection.

```
/65:    close(10)                                       = 0
/65:    lwp_park(0x00000000, 0)         (sleeping...)
```

Finally the worker thread closes the file that it has just delivered and blocks until the listener assigns it another connection.

```
/67:    accept(3, 0x001FEB74, 0x001FEB94, 1) (sleeping...)
```

Meanwhile, the listener thread is able to accept another connection as soon as it has dispatched this connection to a worker thread (subject to some flow-control logic in the worker MPM that throttles the listener if all the available workers are busy). Though it isn't apparent from this trace, the next `accept(2)` can (and usually does, under high load conditions) occur in parallel with the worker thread's handling of the just-accepted connection.

Security Tips

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

Keep up to Date

The Apache HTTP Server has a good record for security and a developer community highly concerned about security issues. But it is inevitable that some problems -- small or large -- will be discovered in software after it is released. For this reason, it is crucial to keep aware of updates to the software. If you have obtained your version of the HTTP Server directly from Apache, we highly recommend you subscribe to the Apache HTTP Server Announcements List where you can keep informed of new releases and security updates. Similar services are available from most third-party distributors of Apache software.

Of course, most times that a web server is compromised, it is not because of problems in the HTTP Server code. Rather, it comes from problems in add-on code, CGI scripts, or the underlying Operating System. You must therefore stay aware of problems and updates with all the software on your system.

Denial of Service (DoS) attacks

All network servers can be subject to denial of service attacks that attempt to prevent responses to clients by tying up the resources of the server. It is not possible to prevent such attacks entirely, but you can do certain things to mitigate the problems that they create.

Often the most effective anti-DoS tool will be a firewall or other operating-system configurations. For example, most firewalls can be configured to restrict the number of simultaneous connections from any individual IP address or network, thus preventing a range of simple attacks. Of course this is no help against Distributed Denial of Service attacks (DDoS).

There are also certain Apache HTTP Server configuration settings that can help mitigate problems:

- The `RequestReadTimeout` directive allows to limit the time a client may take to send the request.
- The `TimeOut` directive should be lowered on sites that are subject to DoS attacks. Setting this to as low as a few seconds may be appropriate. As `TimeOut` is currently used for several different operations, setting it to a low value introduces problems with long running CGI scripts.
- The `KeepAliveTimeout` directive may be also lowered on sites that are subject to DoS attacks. Some sites even turn off the keepalives completely via `KeepAlive`, which has of course other drawbacks on performance.
- The values of various timeout-related directives provided by other modules should be checked.
- The directives `LimitRequestBody`, `LimitRequestFields`, `LimitRequestFieldSize`, `LimitRequestLine`, and `LimitXMLRequestBody` should be carefully configured to limit resource consumption triggered by client input.
- On operating systems that support it, make sure that you use the `AcceptFilter` directive to offload part of the request processing to the operating system. This is active by default in Apache httpd, but may require reconfiguration of your kernel.
- Tune the `MaxRequestWorkers` directive to allow the server to handle the maximum number of simultaneous connections without running out of resources. See also the performance tuning documentation.

- The use of a threaded mpm may allow you to handle more simultaneous connections, thereby mitigating DoS attacks. Further, the `event` mpm uses asynchronous processing to avoid devoting a thread to each connection. Due to the nature of the OpenSSL library the `event` mpm is currently incompatible with `mod_ssl` and other input filters. In these cases it falls back to the behaviour of the `worker` mpm.
- There are a number of third-party modules available through http://modules.apache.org/ that can restrict certain client behaviors and thereby mitigate DoS problems.

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`.

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

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

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

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

CGI in General

First of all, you always have to remember that you must trust the writers of the CGI scripts/programs or your ability to spot potential security holes in CGI, whether they were deliberate or accidental. CGI scripts can run essentially arbitrary commands on your system with the permissions of the web server user and can therefore be extremely dangerous if they are not carefully checked.

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.

Non Script Aliased CGI

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

- You trust your users not to write scripts which will deliberately or accidentally expose your system to an attack.
- You consider security at your site to be so feeble in other areas, as to make one more potential hole irrelevant.
- 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.

Other sources of dynamic content

Embedded scripting options which run as part of the server itself, such as `mod_php`, `mod_perl`, `mod_tcl`, and `mod_python`, run under the identity of the server itself (see the `User` directive), and therefore scripts executed by these engines potentially can access anything the server user can. Some scripting engines may provide restrictions, but it is better to be safe and assume not.

Dynamic content security

When setting up dynamic content, such as `mod_php`, `mod_perl` or `mod_python`, many security considerations get out of the scope of `httpd` itself, and you need to consult documentation from those modules. For example, PHP lets you setup Safe Mode, which is most usually disabled by default. Another example is Suhosin, a PHP addon for more security. For more information about those, consult each project documentation.

At the Apache level, a module named mod_security can be seen as a HTTP firewall and, provided you configure it finely enough, can help you enhance your dynamic content security.

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.

Note that this setting is the default since Apache 2.3.9.

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:

```
# cd /; ln -s / public_html
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 "/">
    Require all denied
</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">
    Require all granted
</Directory>
<Directory "/usr/local/httpd">
    Require all granted
</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. We

strongly recommend that you include the following line in your server configuration files:

```
UserDir disabled root
```

Watching Your Logs

To keep up-to-date with what is actually going on against your server you have to check the Log Files. Even though the log files only reports what has already happened, they will give you some understanding of what attacks is thrown against the server and allow you to check if the necessary level of security is present.

A couple of examples:

```
grep -c "/jsp/source.jsp?/jsp/ /jsp/source.jsp??"
access_log
grep "client denied" error_log | tail -n 10
```

The first example will list the number of attacks trying to exploit the Apache Tomcat Source.JSP Malformed Request Information Disclosure Vulnerability, the second example will list the ten last denied clients, for example:

```
[Thu Jul 11 17:18:39 2002] [error] [client
foo.example.com] client denied by server configuration:
/usr/local/apache/htdocs/.htpasswd
```

As you can see, the log files only report what already has happened, so if the client had been able to access the `.htpasswd` file you would have seen something similar to:

```
foo.example.com - - [12/Jul/2002:01:59:13 +0200] "GET
/.htpasswd HTTP/1.1"
```

in your Access Log. This means you probably commented out the following in your server configuration file:

```
<Files ".ht*">
    Require all denied
</Files>
```

Merging of configuration sections

The merging of configuration sections is complicated and sometimes directive specific. Always test your changes when creating dependencies on how directives are merged.

For modules that don't implement any merging logic, such as `mod_access_compat`, the behavior in later sections depends on whether the later section has any directives from the module. The configuration is inherited until a change is made, at which point the configuration is *replaced* and not merged.

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 Modules	Related Directives
	ServerName
	ServerAdmin
	ServerSignature
	ServerTokens
	UseCanonicalName
	UseCanonicalPhysicalPort

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, UseCanonicalName and UseCanonicalPhysicalPort 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, httpd 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 Modules	Related Directives
	CoreDumpDirectory
	DocumentRoot
	ErrorLog
	Mutex
	PidFile
	ScoreBoardFile
	ServerRoot

These directives control the locations of the various files that httpd 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 Modules	Related Directives
	LimitRequestBody
	LimitRequestFields
	LimitRequestFieldsize
	LimitRequestLine
	RLimitCPU
	RLimitMEM
	RLimitNPROC
	ThreadStackSize

The `LimitRequest*` directives are used to place limits on the amount of resources httpd 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 httpd children. In particular, this will control resources used by CGI scripts and SSI exec commands.

The `ThreadStackSize` directive is used with some platforms to control the stack size.

Implementation Choices

Related Modules	Related Directives
	`Mutex`

The `Mutex` directive can be used to change the underlying implementation used for mutexes, in order to relieve functional or performance problems with APR's default choice.

Apache SSL/TLS 加密

此翻译可能过期。要了解最近的更改，请阅读英文版。

Apache HTTP 服务器模块 mod_ssl 提供了与 OpenSSL 的接口，它使用安全套接字层和传输层安全协议提供了强加密。 此模块与这篇文档都基于 Ralf S. Engelschall 的 mod_ssl 项目。

文档

mod_ssl

此模块提供的指令和环境变量的文档位于 mod_ssl 参考手册。

suEXEC Support

The **suEXEC** feature provides users of the Apache HTTP Server 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.

Before we begin

Before jumping head-first into this document, you should be aware that certain assumptions are made about you and the environment in which you will be using suexec.

First, it is assumed that you are using a UNIX derivative 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 HTTP Server development team for consideration.

Fourth, and last, it has been the decision of the Apache HTTP Server development team to **NOT** make suEXEC part of the default installation of Apache httpd. 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 we hope 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!

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 HTTP 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 httpd 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. **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.

2. **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 HTTP 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 httpd binary.

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 CGI or SSI program have an unsafe hierarchical reference?**

   Does the target CGI or SSI program's path contain a leading '/' or have a '..' backreference? These are not allowed; the target CGI/SSI program must reside within suEXEC's document root (see `--with-suexec-docroot=`*`DIR`* below).

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?**

   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. **Can we change directory to the one in which the target CGI/SSI program resides?**

    If it doesn't exist, it can't very well contain files. If we can't change directory to it, it might as well not exist.

13. **Is the directory within the httpd webspace?**

    If the request is for a regular portion of the server, is the requested directory within suEXEC's document root? If the request is for a `UserDir`, is the requested directory within the directory configured as suEXEC's userdir (see suEXEC's configuration options)?

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 CGI/SSI program exist?**

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

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

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

17. **Is the target CGI/SSI 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's 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 CGI/SSI program and execute?**

    Here is where suEXEC ends and the target CGI/SSI program begins.

This is the standard operation of 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.

Configuring & Installing suEXEC

Here's where we begin the fun.

suEXEC configuration options

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

--enable-suexec-capabilities
: **Linux specific:** Normally, the `suexec` binary is installed "setuid/setgid root", which allows it to run with the full privileges of the root user. If this option is used, the `suexec` binary will instead be installed with only the setuid/setgid "capability" bits set, which is the subset of full root privileges required for suexec operation. Note that the `suexec` binary may not be able to write to a log file in this mode; it is recommended that the `--with-suexec-syslog --without-suexec-logfile` options are used in conjunction with this mode, so that syslog logging is used instead.

--with-suexec-bin=*PATH*
: The path to the `suexec` binary must be hard-coded in the server for security reasons. Use this option to override the default path. *e.g.* `--with-suexec-bin=/usr/sbin/suexec`

--with-suexec-caller=*UID*
: The username under which httpd normally runs. This is the only user allowed to execute the suEXEC wrapper.

--with-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!**

--with-suexec-docroot=*DIR*
: Define as the DocumentRoot set for httpd. This will be the only hierarchy (aside from `UserDir`s) 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.

--with-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.

--with-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.

--with-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`).

--with-suexec-syslog
: If defined, suexec will log notices and errors to syslog instead of a logfile. This option must be combined with `--without-suexec-logfile`.

--with-suexec-safepath=*PATH*

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

Compiling and installing the suEXEC wrapper

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

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. The default location is "/usr/local/apache2/bin/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.

Setting paranoid permissions

Although the suEXEC wrapper will check to ensure that its caller is the correct user as specified with the `--with-suexec-caller` `configure` option, there is always the possibility that a system or library call suEXEC uses before this check may be exploitable on your system. To counter this, and because it is best-practise in general, you should use filesystem permissions to ensure that only the group httpd runs as may execute suEXEC.

If for example, your web server is configured to run as:

```
User www
Group webgroup
```

and `suexec` is installed at "/usr/local/apache2/bin/suexec", you should run:

```
chgrp webgroup /usr/local/apache2/bin/suexec
chmod 4750 /usr/local/apache2/bin/suexec
```

This will ensure that only the group httpd runs as can even execute the suEXEC wrapper.

Enabling & Disabling suEXEC

Upon startup of httpd, it looks for the file `suexec` in the directory defined by the `--sbindir` option (default is "/usr/local/apache/sbin/suexec"). If httpd 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 HTTP Server is already running you must kill and restart httpd. Restarting it with a simple HUP or USR1 signal will not be enough.

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

Using suEXEC

Requests for CGI programs will call the suEXEC wrapper only if they are for a virtual host containing a SuexecUserGroup directive or if they are processed by mod_userdir.

Virtual Hosts:
One way to use the suEXEC wrapper is through the SuexecUserGroup directive in VirtualHost definitions. By setting this directive 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 this directive is not specified for a <VirtualHost> then the main server userid is assumed.

User directories:
Requests that are processed by mod_userdir will call the suEXEC wrapper to execute CGI programs under the userid of the requested user directory. 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. See also the `--with-suexec-userdir` compile time option.

Debugging suEXEC

The suEXEC wrapper will write log information to the file defined with the `--with-suexec-logfile` option as indicated above, or to syslog if `--with-suexec-syslog` is used. If you feel you have configured and installed the wrapper properly, have a look at the log and the error_log for the server to see where you may have gone astray. The output of `"suexec -V"` will show the options used to compile suexec, if using a binary distribution.

Beware the Jabberwock: Warnings & Examples

NOTE! This section may not be complete.

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 httpd 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.

Apache mod_rewrite

此翻译可能过期。要了解最近的更改，请阅读英文版。

mod_rewrite 提供了基于正则表达式规则动态修改传入的请求的 URL 的方法。 这允许你以自己喜欢的任意方法映射任意 URL 到你的内部 URL 结构。

它支持无限的规则，以及为每个规则附加条件，从而提供了一个真正灵活且强大的 URL 操作机制。URL 操作可以依赖于各种测试，例如服务器变量，环境变量，HTTP 头，时戳，甚至外部数据库查询等，以便完成 URL 单元匹配。

这个模块在服务器上下文 (httpd.conf)，虚拟主机上下文 (<VirtualHost> 指令块)，目录上下文 (.htaccess 文件和 <Directory> 指令块) 对完整的 URL (包含目录信息部分和查询字符串部分) 操作。 重写结果可以导致新的规则处理，内部的后续处理，外部请求重定向，甚至透过内部代理， 这取决于你为规则附加的标志。

既然 mod_rewrite 这么强大，它当然是相当复杂。这篇文档作为参考手册的补充，试图减轻一些复杂性， 提供你可能使用 mod_rewrite 的常见场景的有充分注释的例子。 但是，我们也试图告诉你，在什么时候你不应当使用 mod_rewrite， 可以使用其它标准的 Apache 特性来达到目的，以避免无谓的复杂性。

参见

- mod_rewrite 参考手册
- 从 URL 映射到文件系统
- mod_rewrite wiki
- 术语

Apache 虚拟主机文档

此翻译可能过期。要了解最近的更改，请阅读英文版。

术语*虚拟主机*指的是在单一机器上运行多个网站 (例如 company1.example.com 和 company2.example.com) 。 虚拟主机可以“基于 IP”，即每个 IP 一个站点； 或者“基于名称”， 即每个 IP 多个站点。这些站点运行在同一物理服务器上的事实不会明显的透漏给最终用户。

Apache 是第一个支持基于 IP 的虚拟主机的服务器。 Apache 版本 1.1 和更新的版本同时支持基于 IP 和基于名称的虚拟主机。 基于名称的虚拟主机有时候称为*基于主机*或*非 IP* 的虚拟主机.

以下解释是在 Apache 中支持虚拟主机的所有详细信息的文档页面列表。

虚拟主机支持

- 基于名称的虚拟主机 (每个 IP 多个站点)
- 基于 IP 的虚拟主机 (每个 IP 一个站点)
- 虚拟主机样例
- 文件句柄限制 (或者*日志文件太多*)
- 动态配置的大规模虚拟主机
- 虚拟主机匹配的深入讨论

配置指令

- <VirtualHost>
- ServerName
- ServerAlias
- ServerPath

如果你要调试虚拟主机配置，你会发现 Apache 的命令行参数 -S 非常有用。即输入以下命令:

```
/usr/local/apache2/bin/httpd -S
```

这个命令将会显示 Apache 是如何解析配置文件的。仔细检查 IP 地址与服务器名称可能会帮助你发现配置错误 (参见 httpd 程序文档，以便了解其它命令行选项)。

Authentication and Authorization

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.

For general access control, see the Access Control How-To.

Related Modules and Directives

There are three types of modules involved in the authentication and authorization process. You will usually need to choose at least one module from each group.

- Authentication type (see the `AuthType` directive)
  - mod_auth_basic
  - mod_auth_digest
- Authentication provider (see the `AuthBasicProvider` and `AuthDigestProvider` directives)
  - mod_authn_anon
  - mod_authn_dbd
  - mod_authn_dbm
  - mod_authn_file
  - mod_authnz_ldap
  - mod_authn_socache
- Authorization (see the `Require` directive)
  - mod_authnz_ldap
  - mod_authz_dbd
  - mod_authz_dbm
  - mod_authz_groupfile
  - mod_authz_host
  - mod_authz_owner
  - mod_authz_user

In addition to these modules, there are also mod_authn_core and mod_authz_core. These modules implement core directives that are core to all auth modules.

The module mod_authnz_ldap is both an authentication and authorization provider. The module mod_authz_host provides authorization and access control based on hostname, IP address or characteristics of the request, but is not part of the authentication provider system. For backwards compatibility with the mod_access, there is a new module mod_access_compat.

You probably also want to take a look at the Access Control howto, which discusses the various ways to control access to your server.

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.

Note:

If your data really needs to be secure, consider using mod_ssl in addition to any authentication.

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.

You will also need to make sure that the modules mod_authn_core and mod_authz_core have either been built into the httpd binary or loaded by the httpd.conf configuration file. Both of these modules provide core directives and functionality that are critical to the configuration and use of authentication and authorization in the web server.

Getting it working

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

First, you need to create a password file. Exactly how you do this will vary depending on what authentication provider you have chosen. More on that later. To start with, we'll use a text 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 will be located in the bin directory of wherever you installed Apache. If you have installed Apache from a third-party package, it may be in your execution path.

To create the file, type:

```
htpasswd -c /usr/local/apache/passwd/passwords 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. With a default installation, it's located at /usr/local/apache2/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/htdocs/secret"> section.

```
AuthType Basic
AuthName "Restricted Files"
# (Following line optional)
AuthBasicProvider file
AuthUserFile "/usr/local/apache/passwd/passwords"
Require user rbowen
```

Let's examine each of those directives individually. The AuthType directive selects the method that is used to authenticate the user. The most common method is Basic, and this is the method implemented by mod_auth_basic. It is important to be aware, however, that Basic authentication sends the password from the client to the server unencrypted. This method should therefore not be used for highly sensitive data, unless accompanied by mod_ssl. Apache supports one other authentication method: AuthType Digest. This method is implemented by mod_auth_digest and was intended to be more secure. This is no longer the case and the connection should be encrypted with mod_ssl instead.

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 AuthBasicProvider is, in this case, optional, since file is the default value for this directive. You'll need to use this directive if you are choosing a different source for authentication, such as mod_authn_dbm or mod_authn_dbd.

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_authn_dbm module provides the AuthDBMUserFile directive. These files can be created and manipulated with the dbmmanage and htdbm programs. 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/passwords 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"
# Optional line:
AuthBasicProvider file
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 the 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.

Alternate password storage

Because storing passwords in plain text files has the above problems, you may wish to store your passwords somewhere else, such as in a database.

mod_authn_dbm and mod_authn_dbd are two modules which make this possible. Rather than selecting AuthBasicProvider file, instead you can choose dbm or dbd as your storage format.

To select a dbm file rather than a text file, for example:

```
<Directory "/www/docs/private">
    AuthName "Private"
    AuthType Basic
    AuthBasicProvider dbm
    AuthDBMUserFile "/www/passwords/passwd.dbm"
    Require valid-user
</Directory>
```

Other options are available. Consult the mod_authn_dbm documentation for more details.

Using multiple providers

With the introduction of the new provider based authentication and authorization architecture, you are no longer locked into a single authentication or authorization method. In fact any number of the providers can be mixed and matched to provide you with exactly the scheme that meets your needs. In the following example, both the file and LDAP based authentication providers are being used.

```
<Directory "/www/docs/private">
    AuthName "Private"
    AuthType Basic
    AuthBasicProvider file ldap
    AuthUserFile "/usr/local/apache/passwd/passwords"
    AuthLDAPURL ldap://ldaphost/o=yourorg
    Require valid-user
</Directory>
```

In this example the file provider will attempt to authenticate the user first. If it is unable to authenticate the user, the LDAP provider will be called. This allows the scope of authentication to be broadened if your organization implements more than one type of authentication store. Other authentication and authorization scenarios may include mixing one type of authentication with a different type of authorization. For example, authenticating against a password file yet authorizing against an LDAP directory.

Just as multiple authentication providers can be implemented, multiple authorization methods can also be used. In this example both file group authorization as well as LDAP group authorization is being used.

```
<Directory "/www/docs/private">
    AuthName "Private"
    AuthType Basic
    AuthBasicProvider file
    AuthUserFile "/usr/local/apache/passwd/passwords"
    AuthLDAPURL ldap://ldaphost/o=yourorg
    AuthGroupFile "/usr/local/apache/passwd/groups"
    Require group GroupName
    Require ldap-group cn=mygroup,o=yourorg
</Directory>
```

To take authorization a little further, authorization container directives such as <RequireAll> and <RequireAny> allow logic to be applied so that the order in

which authorization is handled can be completely controlled through the configuration. See Authorization Containers for an example of how they may be applied.

Beyond just authorization

The way that authorization can be applied is now much more flexible than just a single check against a single data store. Ordering, logic and choosing how authorization will be done is now possible.

Applying logic and ordering

Controlling how and in what order authorization will be applied has been a bit of a mystery in the past. In Apache 2.2 a provider-based authentication mechanism was introduced to decouple the actual authentication process from authorization and supporting functionality. One of the side benefits was that authentication providers could be configured and called in a specific order which didn't depend on the load order of the auth module itself. This same provider based mechanism has been brought forward into authorization as well. What this means is that the `Require` directive not only specifies which authorization methods should be used, it also specifies the order in which they are called. Multiple authorization methods are called in the same order in which the `Require` directives appear in the configuration.

With the introduction of authorization container directives such as `<RequireAll>` and `<RequireAny>`, the configuration also has control over when the authorization methods are called and what criteria determines when access is granted. See Authorization Containers for an example of how they may be used to express complex authorization logic.

By default all `Require` directives are handled as though contained within a `<RequireAny>` container directive. In other words, if any of the specified authorization methods succeed, then authorization is granted.

Using authorization providers for access control

Authentication by username and password is only part of the story. Frequently you want to let people in based on something other than who they are. Something such as where they are coming from.

The authorization providers `all`, `env`, `host` and `ip` let you allow or deny access based on other host based criteria such as host name or ip address of the machine requesting a document.

The usage of these providers is specified through the `Require` directive. This directive registers the authorization providers that will be called during the authorization stage of the request processing. For example:

```
Require ip address
```

where *address* is an IP address (or a partial IP address) or:

```
Require host domain_name
```

where *domain_name* is 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:

```
<RequireAll>
    Require all granted
    Require not ip 10.252.46.165
</RequireAll>
```

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.

```
<RequireAll>
    Require all granted
    Require not host host.example.com
</RequireAll>
```

And, if you'd like to block access from an entire domain, you can specify just part of an address or domain name:

```
<RequireAll>
    Require all granted
    Require not ip 192.168.205
    Require not host phishers.example.com moreidiots.ex
    Require not host ke
</RequireAll>
```

Using `<RequireAll>` with multiple `<Require>` directives, each negated with `not`, will only allow access, if all of negated conditions are true. In other words, access will be blocked, if any of the negated conditions fails.

Access Control backwards compatibility

One of the side effects of adopting a provider based mechanism for authentication is that the previous access control directives `Order`, `Allow`, `Deny` and `Satisfy` are no longer needed. However to provide backwards compatibility for older configurations, these directives have been moved to the `mod_access_compat` module.

> **Note**
>
> The directives provided by `mod_access_compat` have been deprecated by `mod_authz_host`. Mixing old directives like `Order`, `Allow` or `Deny` with new ones like `Require` is technically possible but discouraged. The `mod_access_compat` module was created to support configurations containing only old directives to facilitate the 2.4 upgrade. Please check the upgrading guide for more information.

Authentication Caching

There may be times when authentication puts an unacceptable load on a provider or on your network. This is most likely to affect users of `mod_authn_dbd` (or third-party/custom providers). To deal with this, HTTPD 2.3/2.4 introduces a new caching provider `mod_authn_socache` to cache credentials and reduce the load on the origin provider(s).

This may offer a substantial performance boost to some users.

More information

You should also read the documentation for `mod_auth_basic` and `mod_authz_host` which contain some more information about how this all works. The directive `<AuthnProviderAlias>` can also help in simplifying certain authentication configurations.

The various ciphers supported by Apache for authentication data are explained in Password Encryptions.

And you may want to look at the Access Control howto, which discusses a number of related topics.

Apache Tutorial: Dynamic Content with CGI

Introduction

Related Modules	Related Directives
mod_alias	AddHandler
mod_cgi	Options
mod_cgid	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 a simple way to put dynamic content on your web site, using whatever programming language you're most familiar with. 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.

Note: If Apache has been built with shared module support you need to ensure that the module is loaded; in your `httpd.conf` you need to make sure the `LoadModule` directive has not been commented out. A correctly configured directive may look like this:

```
LoadModule cgid_module modules/mod_cgid.so
```

On Windows, or using a non-threaded MPM like prefork, A correctly configured directive may look like this:

```
LoadModule cgi_module modules/mod_cgi.so
```

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/apache2/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/apache2/cgi-bin/`, and should be treated as a CGI program.

For example, if the URL `http://www.example.com/cgi-bin/test.pl` is requested, Apache will attempt to execute the file `/usr/local/apache2/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.

There are two steps to allowing CGI execution in an arbitrary directory. First, the cgi-script handler must be activated using the AddHandler or SetHandler directive. Second, ExecCGI must be specified in the Options directive.

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/apache2/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

The .htaccess tutorial shows how to activate CGI programs if you do not have access to httpd.conf.

User Directories

To allow CGI program execution for any file ending in .cgi in users' directories, you can use the following configuration.

```
<Directory "/home/*/public_html">
    Options +ExecCGI
    AddHandler cgi-script .cgi
</Directory>
```

If you wish designate a cgi-bin subdirectory of a user's directory where everything will be treated as a CGI program, you can use the following.

```
<Directory "/home/*/public_html/cgi-bin">
    Options ExecCGI
    SetHandler cgi-script
</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\n\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. If the output is correct, but the browser is not processing it correctly, make sure you have the correct `Content-Type` set in your CGI program.

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.

Path information and environment

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 may not have the same `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.

> When editing CGI scripts on Windows, end-of-line characters may be appended to the interpreter path. Ensure that files are then transferred to the server in ASCII mode. Failure to do so may result in "Command not found" warnings from the OS, due to the unrecognized end-of-line character being interpreted as a part of the interpreter filename.

Missing environment variables

If your CGI program depends on non-standard environment variables, you will need to assure that those variables are passed by Apache.

When you miss HTTP headers from the environment, make sure they are formatted according to RFC 2616, section 4.2: Header names must start with a letter, followed only by letters, numbers or hyphen. Any header violating this rule will be dropped silently.

Program 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. The first thing to do is to make sure that your program runs from the command line before testing it via the web server. For example, try:

```
cd /usr/local/apache2/cgi-bin
./first.pl
```

(Do not call the `perl` interpreter. The shell and Apache should find the interpreter using the path information on the first line of the script.)

The first thing you see written by your program should be a set of HTTP headers, including the `Content-Type`, followed by a blank line. If you see anything else, Apache will return the `Premature end of script headers` error if you try to run it through the server. See Writing a CGI program above for more details.

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.

Suexec

The suexec support 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 `Premature end of script headers`.

To check if you are using suexec, run `apachectl -V` and check for the location of `SUEXEC_BIN`. If Apache finds an `suexec` binary there on startup, suexec will be activated.

Unless you fully understand suexec, you should not be using it. To disable suexec, simply remove (or rename) the `suexec` binary pointed to by `SUEXEC_BIN` and then restart the server. If, after reading about suexec, you still wish to use it, then run `suexec -V` to find the location of the suexec log file, and use that log file to find what policy you are violating.

What's going on behind the scenes?

As you become more advanced in CGI programming, it will become useful to understand more about what's happening behind the scenes. Specifically, how the browser and server communicate with one another. Because although it's all very well to write a program that prints "Hello, World.", it's not particularly useful.

Environment variables

Environment variables are values that float around you as you use your computer. They are useful things like your path (where the computer searches for the actual file implementing a command when you type it), your username, your terminal type, and so on. For a full list of your normal, every day environment variables, type `env` at a command prompt.

During the CGI transaction, the server and the browser also set environment variables, so that they can communicate with one another. These are things like the browser type (Netscape, IE, Lynx), the server type (Apache, IIS, WebSite), the name of the CGI program that is being run, and so on.

These variables are available to the CGI programmer, and are half of the story of the client-server communication. The complete list of required variables is at Common Gateway Interface RFC.

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
use strict;
```

```
use warnings;

print "Content-type: text/html\n\n";
foreach my $key (keys %ENV) {
    print "$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 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 https://web.mit.edu/wwwdev/www/cgic.html.

For more information

The current CGI specification is available in the Common Gateway Interface RFC.

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 Tutorial: .htaccess files

`.htaccess` files provide a way to make configuration changes on a per-directory basis.

.htaccess files

Related Modules	Related Directives
core	AccessFileName
mod_authn_file	AllowOverride
mod_authz_groupfile	Options
mod_cgi	AddHandler
mod_include	SetHandler
mod_mime	AuthType
	AuthName
	AuthUserFile
	AuthGroupFile
	Require

You should avoid using `.htaccess` files completely if you have access to httpd main server config file. Using `.htaccess` files slows down your Apache http server. Any directive that you can include in a `.htaccess` file is better set in a `Directory` block, as it will have the same effect with better performance.

What they are/How to use them

`.htaccess` files (or "distributed configuration files") provide a way to make configuration changes on a per-directory basis. A file, containing one or more configuration directives, is placed in a particular document directory, and the directives apply to that directory, and all subdirectories thereof.

Note:

If you want to call your `.htaccess` file something else, you can change the name of the file using the `AccessFileName` directive. For example, if you would rather call the file `.config` then you can put the following in your server configuration file:

```
AccessFileName ".config"
```

In general, `.htaccess` files use the same syntax as the main configuration files. What you can put in these files is determined by the `AllowOverride` directive. This directive specifies, in categories, what directives will be honored if they are found in a `.htaccess` file. If a directive is permitted in a `.htaccess` file, the documentation for that directive will contain an Override section, specifying what value must be in `AllowOverride` in order for that directive to be permitted.

For example, if you look at the documentation for the `AddDefaultCharset` directive, you will find that it is permitted in `.htaccess` files. (See the Context line in the directive summary.) The Override line reads `FileInfo`. Thus, you must have at least `AllowOverride FileInfo` in order for this directive to be honored in `.htaccess` files.

Example:

Context:	server config, virtual host, directory, .htaccess

Override: FileInfo

If you are unsure whether a particular directive is permitted in a `.htaccess` file, look at the documentation for that directive, and check the Context line for ".htaccess".

When (not) to use .htaccess files

In general, you should only use `.htaccess` files when you don't have access to the main server configuration file. There is, for example, a common misconception that user authentication should always be done in `.htaccess` files, and, in more recent years, another misconception that `mod_rewrite` directives must go in `.htaccess` files. This is simply not the case. You can put user authentication configurations in the main server configuration, and this is, in fact, the preferred way to do things. Likewise, `mod_rewrite` directives work better, in many respects, in the main server configuration.

`.htaccess` files should be used in a case where the content providers need to make configuration changes to the server on a per-directory basis, but do not have root access on the server system. In the event that the server administrator is not willing to make frequent configuration changes, it might be desirable to permit individual users to make these changes in `.htaccess` files for themselves. This is particularly true, for example, in cases where ISPs are hosting multiple user sites on a single machine, and want their users to be able to alter their configuration.

However, in general, use of `.htaccess` files should be avoided when possible. Any configuration that you would consider putting in a `.htaccess` file, can just as effectively be made in a `<Directory>` section in your main server configuration file.

There are two main reasons to avoid the use of `.htaccess` files.

The first of these is performance. When `AllowOverride` is set to allow the use of `.htaccess` files, httpd will look in every directory for `.htaccess` files. Thus, permitting `.htaccess` files causes a performance hit, whether or not you actually even use them! Also, the `.htaccess` file is loaded every time a document is requested.

Further note that httpd must look for `.htaccess` files in all higher-level directories, in order to have a full complement of directives that it must apply. (See section on how directives are applied.) Thus, if a file is requested out of a directory `/www/htdocs/example`, httpd must look for the following files:

```
/.htaccess
/www/.htaccess
/www/htdocs/.htaccess
/www/htdocs/example/.htaccess
```

And so, for each file access out of that directory, there are 4 additional file-system accesses, even if none of those files are present. (Note that this would only be the case if `.htaccess` files were enabled for `/`, which is not usually the case.)

In the case of `RewriteRule` directives, in `.htaccess` context these regular expressions must be re-compiled with every request to the directory, whereas in main server configuration context they are compiled once and cached. Additionally, the rules themselves are more complicated, as one must work around the restrictions that come with per-directory context and `mod_rewrite`. Consult the Rewrite Guide for more detail on this subject.

The second consideration is one of security. You are permitting users to modify server configuration, which may result in changes over which you have no

control. Carefully consider whether you want to give your users this privilege. Note also that giving users less privileges than they need will lead to additional technical support requests. Make sure you clearly tell your users what level of privileges you have given them. Specifying exactly what you have set `AllowOverride` to, and pointing them to the relevant documentation, will save yourself a lot of confusion later.

Note that it is completely equivalent to put a `.htaccess` file in a directory `/www/htdocs/example` containing a directive, and to put that same directive in a Directory section `<Directory "/www/htdocs/example">` in your main server configuration:

`.htaccess` file in `/www/htdocs/example`:

Contents of .htaccess file in `/www/htdocs/example`

```
AddType text/example ".exm"
```

Section from your `httpd.conf` file

```
<Directory "/www/htdocs/example">
    AddType text/example ".exm"
</Directory>
```

However, putting this configuration in your server configuration file will result in less of a performance hit, as the configuration is loaded once when httpd starts, rather than every time a file is requested.

The use of `.htaccess` files can be disabled completely by setting the `AllowOverride` directive to `none`:

```
AllowOverride None
```

How directives are applied

The configuration directives found in a `.htaccess` file are applied to the directory in which the `.htaccess` file is found, and to all subdirectories thereof. However, it is important to also remember that there may have been `.htaccess` files in directories higher up. Directives are applied in the order that they are found. Therefore, a `.htaccess` file in a particular directory may override directives found in `.htaccess` files found higher up in the directory tree. And those, in turn, may have overridden directives found yet higher up, or in the main server configuration file itself.

Example:

In the directory `/www/htdocs/example1` we have a `.htaccess` file containing the following:

```
Options +ExecCGI
```

(Note: you must have "`AllowOverride Options`" in effect to permit the use of the "`Options`" directive in `.htaccess` files.)

In the directory `/www/htdocs/example1/example2` we have a `.htaccess` file containing:

```
Options Includes
```

Because of this second `.htaccess` file, in the directory `/www/htdocs/example1/example2`, CGI execution is not permitted, as only

`Options Includes` is in effect, which completely overrides any earlier setting that may have been in place.

Merging of .htaccess with the main configuration files

As discussed in the documentation on Configuration Sections, `.htaccess` files can override the `<Directory>` sections for the corresponding directory, but will be overridden by other types of configuration sections from the main configuration files. This fact can be used to enforce certain configurations, even in the presence of a liberal `AllowOverride` setting. For example, to prevent script execution while allowing anything else to be set in `.htaccess` you can use:

```
<Directory "/www/htdocs">
    AllowOverride All
</Directory>

<Location "/">
    Options +IncludesNoExec -ExecCGI
</Location>
```

This example assumes that your `DocumentRoot` is `/www/htdocs`.

Authentication example

If you jumped directly to this part of the document to find out how to do authentication, it is important to note one thing. There is a common misconception that you are required to use `.htaccess` files in order to implement password authentication. This is not the case. Putting authentication directives in a `<Directory>` section, in your main server configuration file, is the preferred way to implement this, and `.htaccess` files should be used only if you don't have access to the main server configuration file. See above for a discussion of when you should and should not use `.htaccess` files.

Having said that, if you still think you need to use a `.htaccess` file, you may find that a configuration such as what follows may work for you.

`.htaccess` file contents:

```
AuthType Basic
AuthName "Password Required"
AuthUserFile "/www/passwords/password.file"
AuthGroupFile "/www/passwords/group.file"
Require group admins
```

Note that `AllowOverride AuthConfig` must be in effect for these directives to have any effect.

Please see the authentication tutorial for a more complete discussion of authentication and authorization.

Server Side Includes example

Another common use of `.htaccess` files is to enable Server Side Includes for a particular directory. This may be done with the following configuration directives, placed in a `.htaccess` file in the desired directory:

```
Options +Includes
AddType text/html "shtml"
AddHandler server-parsed shtml
```

Note that `AllowOverride Options` and `AllowOverride FileInfo` must both be in effect for these directives to have any effect.

Please see the SSI tutorial for a more complete discussion of server-side includes.

Rewrite Rules in .htaccess files

When using `RewriteRule` in `.htaccess` files, be aware that the per-directory context changes things a bit. In particular, rules are taken to be relative to the current directory, rather than being the original requested URI. Consider the following examples:

```
# In httpd.conf
RewriteRule "^/images/(.+)\.jpg" "/images/$1.png"

# In .htaccess in root dir
RewriteRule "^images/(.+)\.jpg" "images/$1.png"

# In .htaccess in images/
RewriteRule "^(.+)\.jpg" "$1.png"
```

In a `.htaccess` in your document directory, the leading slash is removed from the value supplied to `RewriteRule`, and in the `images` subdirectory, `/images/` is removed from it. Thus, your regular expression needs to omit that portion as well.

Consult the mod_rewrite documentation for further details on using `mod_rewrite`.

CGI example

Finally, you may wish to use a `.htaccess` file to permit the execution of CGI programs in a particular directory. This may be implemented with the following configuration:

```
Options +ExecCGI
AddHandler cgi-script "cgi" "pl"
```

Alternately, if you wish to have all files in the given directory be considered to be CGI programs, this may be done with the following configuration:

```
Options +ExecCGI
SetHandler cgi-script
```

Note that `AllowOverride Options` and `AllowOverride FileInfo` must both be in effect for these directives to have any effect.

Please see the CGI tutorial for a more complete discussion of CGI programming and configuration.

Troubleshooting

When you put configuration directives in a `.htaccess` file, and you don't get the desired effect, there are a number of things that may be going wrong.

Most commonly, the problem is that `AllowOverride` is not set such that your configuration directives are being honored. Make sure that you don't have a `AllowOverride None` in effect for the file scope in question. A good test for this is to put garbage in your `.htaccess` file and reload the page. If a server error is not generated, then you almost certainly have `AllowOverride None` in effect.

If, on the other hand, you are getting server errors when trying to access documents, check your httpd error log. It will likely tell you that the directive used in your `.htaccess` file is not permitted.

```
[Fri Sep 17 18:43:16 2010] [alert] [client
192.168.200.51] /var/www/html/.htaccess: DirectoryIndex
not allowed here
```

This will indicate either that you've used a directive that is never permitted in `.htaccess` files, or that you simply don't have `AllowOverride` set to a level sufficient for the directive you've used. Consult the documentation for that particular directive to determine which is the case.

Alternately, it may tell you that you had a syntax error in your usage of the directive itself.

```
[Sat Aug 09 16:22:34 2008] [alert] [client
192.168.200.51] /var/www/html/.htaccess: RewriteCond: bad
flag delimiters
```

In this case, the error message should be specific to the particular syntax error that you have committed.

Apache httpd Tutorial: Introduction to Server Side Includes

Server-side includes provide a means to add dynamic content to existing HTML documents.

Introduction

Related Modules	Related Directives
mod_include	Options
mod_cgi	XBitHack
mod_expires	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.

For example, you might place a directive into an existing HTML page, such as:

```
<!--#echo var="DATE_LOCAL" -->
```

And, when the page is served, this fragment will be evaluated and replaced with its value:

```
Tuesday, 15-Jan-2013 19:28:54 EST
```

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 - shown above. 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:

```
<!--#function 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 function 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` function 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` function.

If you don't like the format in which the date gets printed, you can use the `config` function, 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 function 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` function 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` function:

```
<!--#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

Here's something else that you can do with the `exec` function. 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.

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 environment variables or the variables discussed above (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.

For example, if you wish to customize the text on your web page based on the time of day, you could use the following recipe, placed in the HTML page:

```
Good <!--#if expr="%{TIME_HOUR} <12" -->
morning!
<!--#else -->
afternoon!
<!--#endif -->
```

Any other variable (either ones that you define, or normal environment variables) can be used in conditional statements. See Expressions in Apache HTTP Server for more information on the expression evaluation engine.

With Apache's ability to set environment variables with the `SetEnvIf` directives, and other related directives, this functionality can let you do a wide variety of dynamic content on the server side without resorting a full web application.

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.

Per-user web directories

On systems with multiple users, each user can be permitted to have a web site in their home directory using the UserDir directive. Visitors to a URL http://example.com/~username/ will get content out of the home directory of the user "username", out of the subdirectory specified by the UserDir directive.

Note that, by default, access to these directories is **not** enabled. You can enable access when using UserDir by uncommenting the line:

```
#Include conf/extra/httpd-userdir.conf
```

in the default config file conf/httpd.conf, and adapting the httpd-userdir.conf file as necessary, or by including the appropriate directives in a <Directory> block within the main config file.

Per-user web directories

Related Modules	Related Directives
mod_userdir	UserDir
	DirectoryMatch
	AllowOverride

Setting the file path with UserDir

The UserDir directive specifies a directory out of which per-user content is loaded. This directive may take several different forms.

If a path is given which does not start with a leading slash, it is assumed to be a directory path relative to the home directory of the specified user. Given this configuration:

```
UserDir public_html
```

the URL http://example.com/~rbowen/file.html will be translated to the file path /home/rbowen/public_html/file.html

If a path is given starting with a slash, a directory path will be constructed using that path, plus the username specified. Given this configuration:

```
UserDir /var/html
```

the URL http://example.com/~rbowen/file.html will be translated to the file path /var/html/rbowen/file.html

If a path is provided which contains an asterisk (*), a path is used in which the asterisk is replaced with the username. Given this configuration:

```
UserDir /var/www/*/docs
```

the URL http://example.com/~rbowen/file.html will be translated to the file path /var/www/rbowen/docs/file.html

Multiple directories or directory paths can also be set.

```
UserDir public_html /var/html
```

For the URL `http://example.com/~rbowen/file.html`, Apache will search for `~rbowen`. If it isn't found, Apache will search for `rbowen` in `/var/html`. If found, the above URL will then be translated to the file path `/var/html/rbowen/file.html`

Redirecting to external URLs

The `UserDir` directive can be used to redirect user directory requests to external URLs.

```
UserDir http://example.org/users/*/
```

The above example will redirect a request for `http://example.com/~bob/abc.html` to `http://example.org/users/bob/abc.html`.

Restricting what users are permitted to use this feature

Using the syntax shown in the UserDir documentation, you can restrict what users are permitted to use this functionality:

```
UserDir disabled root jro fish
```

The configuration above will enable the feature for all users except for those listed in the `disabled` statement. You can, likewise, disable the feature for all but a few users by using a configuration like the following:

```
UserDir disabled
UserDir enabled rbowen krietz
```

See `UserDir` documentation for additional examples.

Enabling a cgi directory for each user

In order to give each user their own cgi-bin directory, you can use a `<Directory>` directive to make a particular subdirectory of a user's home directory cgi-enabled.

```
<Directory "/home/*/public_html/cgi-bin/">
    Options ExecCGI
    SetHandler cgi-script
</Directory>
```

Then, presuming that `UserDir` is set to `public_html`, a cgi program `example.cgi` could be loaded from that directory as:

```
http://example.com/~rbowen/cgi-bin/example.cgi
```

Allowing users to alter configuration

If you want to allows users to modify the server configuration in their web space, they will need to use `.htaccess` files to make these changes. Ensure that you have set `AllowOverride` to a value sufficient for the directives that you want to permit the users to modify. See the .htaccess tutorial for additional details on how this works.

Using Apache HTTP Server on Microsoft Windows

This document explains how to install, configure and run Apache 2.5 under Microsoft Windows. If you have questions after reviewing the documentation (and any event and error logs), you should consult the peer-supported users' mailing list.

This document assumes that you are installing a binary distribution of Apache. If you want to compile Apache yourself (possibly to help with development or tracking down bugs), see Compiling Apache for Microsoft Windows.

Operating System Requirements

The primary Windows platform for running Apache 2.5 is Windows 2000 or later. Always obtain and install the current service pack to avoid operating system bugs.

> Apache HTTP Server versions later than 2.2 will not run on any operating system earlier than Windows 2000.

Downloading Apache for Windows

The Apache HTTP Server Project itself does not provide binary releases of software, only source code. Individual committers *may* provide binary packages as a convenience, but it is not a release deliverable.

If you cannot compile the Apache HTTP Server yourself, you can obtain a binary package from numerous binary distributions available on the Internet.

Popular options for deploying Apache httpd, and, optionally, PHP and MySQL, on Microsoft Windows, include:

- ApacheHaus
- Apache Lounge
- Bitnami WAMP Stack
- WampServer
- XAMPP

Customizing Apache for Windows

Apache is configured by the files in the `conf` subdirectory. These are the same files used to configure the Unix version, but there are a few different directives for Apache on Windows. See the directive index 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 can on Unix. Instead there are usually only two Apache processes running: a parent process, and a child which handles the requests. Within the child process each request is handled by a separate thread.

  The process management directives are also different:

  `MaxConnectionsPerChild`: Like the Unix directive, this controls how many connections a single child process will serve before exiting. However, unlike on Unix, a replacement process is not instantly available. Use the default `MaxConnectionsPerChild 0`, unless instructed to change the behavior to overcome a memory leak in third party modules or in-process applications.

Warning: The server configuration file is reread when a 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. It tells the server how many threads it should use. This is the maximum number of connections the server can handle at once, so be sure to set this number high enough for your site if you get a lot of hits. The recommended default is `ThreadsPerChild 150`, but this must be adjusted to reflect the greatest anticipated number of simultaneous connections to accept.

- The directives that accept filenames as arguments must use Windows filenames instead of Unix ones. However, because Apache may interpret backslashes as an "escape character" sequence, you should consistently use forward slashes in path names, not backslashes.

- While filenames are generally case-insensitive on Windows, URLs are still treated internally as case-sensitive before they are mapped to the filesystem. For example, the `<Location>`, `Alias`, and `ProxyPass` directives all use case-sensitive arguments. For this reason, it is particularly important to use the `<Directory>` directive when attempting to limit access to content in the filesystem, since this directive applies to any content in a directory, regardless of how it is accessed. If you wish to assure that only lowercase is used in URLs, you can use something like:

```
RewriteEngine On
RewriteMap lowercase "int:tolower"
RewriteCond "%{REQUEST_URI}" "[A-Z]"
RewriteRule "(.*)"           "${lowercase:$1}" [R,
```

- When running, Apache needs write access only to the logs directory and any configured cache directory tree. Due to the issue of case insensitive and short 8.3 format names, Apache must validate all path names given. This means that each directory which Apache evaluates, from the drive root up to the directory leaf, must have read, list and traverse directory permissions. If Apache2.5 is installed at C:\Program Files, then the root directory, Program Files and Apache2.5 must all be visible to Apache.

- 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 `\Apache2.5\modules` directory. To activate these or other modules, the `LoadModule` directive must be used. For example, to activate the status module, use the following (in addition to the status-activating directives in `access.conf`):

```
LoadModule status_module "modules/mod_status.so"
```

  Information on creating loadable modules is also available.

- Apache can also load ISAPI (Internet Server Application Programming Interface) extensions such as those used by Microsoft IIS and other Windows servers. More information is available. Note that Apache **cannot** load ISAPI Filters, and ISAPI Handlers with some Microsoft feature extensions will not work.

- 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` in Windows, you may find it useful to change the name of this per-directory configuration file using the `AccessFilename` directive.

- Any errors during Apache startup are logged into the Windows event log when running on Windows NT. This mechanism acts as a backup for those situations where Apache is not yet prepared to use the `error.log` file. You can review the Windows Application Event Log by using the Event Viewer, e.g. Start - Settings - Control Panel - Administrative Tools - Event Viewer.

Running Apache as a Service

Apache comes with a utility called the Apache Service Monitor. With it you can see and manage the state of all installed Apache services on any machine on your network. To be able to manage an Apache service with the monitor, you have to first install the service (either automatically via the installation or manually).

You can install Apache as a Windows NT service as follows from the command prompt at the Apache `bin` subdirectory:

```
httpd.exe -k install
```

If you need to specify the name of the service you want to install, use the following command. You have to do this if you have several different service installations of Apache on your computer. If you specify a name during the install, you have to also specify it during any other -k operation.

```
httpd.exe -k install -n "MyServiceName"
```

If you need to have specifically named configuration files for different services, you must use this:

```
httpd.exe -k install -n "MyServiceName" -f
"c:\files\my.conf"
```

If you use the first command without any special parameters except `-k install`, the service will be called `Apache2.5` and the configuration will be assumed to be `conf\httpd.conf`.

Removing an Apache service is easy. Just use:

```
httpd.exe -k uninstall
```

The specific Apache service to be uninstalled can be specified by using:

```
httpd.exe -k uninstall -n "MyServiceName"
```

Normal starting, restarting and shutting down of an Apache service is usually done via the Apache Service Monitor, by using commands like `NET START Apache2.5` and `NET STOP Apache2.5` or via normal Windows service management. Before starting Apache as a service by any means, you should test the service's configuration file by using:

```
httpd.exe -n "MyServiceName" -t
```

You can control an Apache service by its command line switches, too. To start an installed Apache service you'll use this:

```
httpd.exe -k start -n "MyServiceName"
```

To stop an Apache service via the command line switches, use this:

```
httpd.exe -k stop -n "MyServiceName"
```

or

```
httpd.exe -k shutdown -n "MyServiceName"
```

You can also restart a running service and force it to reread its configuration file by using:

```
httpd.exe -k restart -n "MyServiceName"
```

By default, all Apache services are registered to run as the system user (the `LocalSystem` account). The `LocalSystem` account has no privileges to your network via any Windows-secured mechanism, including the file system, named pipes, DCOM, or secure RPC. It has, however, wide privileges locally.

Never grant any network privileges to the `LocalSystem` account! If you need Apache to be able to access network resources, create a separate account for Apache as noted below.

It is recommended that users create a separate account for running Apache service(s). If you have to access network resources via Apache, this is required.

1. Create a normal domain user account, and be sure to memorize its password.
2. Grant the newly-created user a privilege of `Log on as a service` and `Act as part of the operating system`. On Windows NT 4.0 these privileges are granted via User Manager for Domains, but on Windows 2000 and XP you probably want to use Group Policy for propagating these settings. You can also manually set these via the Local Security Policy MMC snap-in.
3. Confirm that the created account is a member of the Users group.
4. Grant the account read and execute (RX) rights to all document and script folders (`htdocs` and `cgi-bin` for example).
5. Grant the account change (RWXD) rights to the Apache `logs` directory.
6. Grant the account read and execute (RX) rights to the `httpd.exe` binary executable.

It is usually a good practice to grant the user the Apache service runs as read and execute (RX) access to the whole Apache2.5 directory, except the `logs` subdirectory, where the user has to have at least change (RWXD) rights.

If you allow the account to log in as a user and as a service, then you can log on with that account 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.

Error code 2186 is a good indication that you need to review the "Log On As" configuration for the service, since Apache cannot access a required network resource. Also, pay close attention to the privileges of the user Apache is configured to run as.

When starting Apache as a service you may encounter an error message from the Windows Service Control Manager. For example, if you try to start Apache by

using the Services applet in the Windows Control Panel, you may get the following message:

```
Could not start the Apache2.5 service on \\COMPUTER
Error 1067; The process terminated unexpectedly.
```

You will get this generic error if there is any problem with starting the Apache service. In order to see what is really causing the problem you should follow the instructions for Running Apache for Windows from the Command Prompt.

If you are having problems with the service, it is suggested you follow the instructions below to try starting httpd.exe from a console window, and work out the errors before struggling to start it as a service again.

Running Apache as a Console Application

Running Apache as a service is usually the recommended way to use it, but it is sometimes easier to work from the command line, especially during initial configuration and testing.

To run Apache from the command line as a console application, use the following command:

```
httpd.exe
```

Apache will execute, and will remain running until it is stopped by pressing Control-C.

You can also run Apache via the shortcut Start Apache in Console placed to `Start Menu --> Programs --> Apache HTTP Server 2.5.xx --> Control Apache Server` during the installation. This will open a console window and start Apache inside it. If you don't have Apache installed as a service, the window will remain visible until you stop Apache by pressing Control-C in the console window where Apache is running in. The server will exit in a few seconds. However, if you do have Apache installed as a service, the shortcut starts the service. If the Apache service is running already, the shortcut doesn't do anything.

If Apache is running as a service, you can tell it to stop by opening another console window and entering:

```
httpd.exe -k shutdown
```

Running as a service should be preferred over running in a console window because this lets Apache end any current operations and clean up gracefully.

But if the server is running in a console window, you can only stop it by pressing Control-C in the same window.

You can also tell Apache to restart. This forces it to reread the configuration file. Any operations in progress are allowed to complete without interruption. To restart Apache, either press Control-Break in the console window you used for starting Apache, or enter

```
httpd.exe -k restart
```

if the server is running as a service.

> 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.

If the Apache console window closes immediately or unexpectedly after startup, open the Command Prompt from the Start Menu --> Programs. Change to the folder to which you installed Apache, type the command `httpd.exe`, and read the error message. Then change to the logs folder, and review the `error.log` file for configuration mistakes. Assuming httpd was installed into `C:\Program Files\Apache Software Foundation\Apache2.5\`, you can do the following:

```
c:
cd "\Program Files\Apache Software
Foundation\Apache2.5\bin"
httpd.exe
```

Then wait for Apache to stop, or press Control-C. Then enter the following:

```
cd ..\logs
more < error.log
```

When working with Apache it is important to know how it will find the configuration file. You can specify a configuration file on the command line in two ways:

- `-f` specifies an absolute or relative path to a particular configuration file:

  ```
  httpd.exe -f "c:\my server files\anotherconfig.conf"
  ```

  or

  ```
  httpd.exe -f files\anotherconfig.conf
  ```

- `-n` specifies the installed Apache service whose configuration file is to be used:

  ```
  httpd.exe -n "MyServiceName"
  ```

In both of these cases, the proper `ServerRoot` should be set in the configuration file.

If you don't specify a configuration file with `-f` or `-n`, Apache will use the file name compiled into the server, such as `conf\httpd.conf`. This built-in path is relative to the installation directory. You can verify the compiled file name from a value labelled as `SERVER_CONFIG_FILE` when invoking Apache with the `-V` switch, like this:

```
httpd.exe -V
```

Apache will then try to determine its `ServerRoot` by trying the following, in this order:

1. A `ServerRoot` directive via the `-C` command line switch.
2. The `-d` switch on the command line.
3. Current working directory.
4. A registry entry which was created if you did a binary installation.
5. The server root compiled into the server. This is `/apache` by default, you can verify it by using `httpd.exe -V` and looking for a value labelled as `HTTPD_ROOT`.

If you did not do a binary install, Apache will in some scenarios complain about the missing registry key. This warning can be ignored if the server was otherwise able to find its configuration file.

The value of this key is the ServerRoot directory which contains the conf subdirectory. When Apache starts it reads the httpd.conf file from that directory. If this file contains a ServerRoot directive which contains a different directory from the one 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 directive in the httpd.conf file to reflect the new location.

Testing the Installation

After starting Apache (either in a console window or as a service) it will be listening on port 80 (unless you changed the Listen directive in the configuration files or installed Apache only for the current user). To connect to the server and access the default page, launch a browser and enter this URL:

```
http://localhost/
```

Apache should respond with a welcome page and you should see "It Works!". If nothing happens or you get an error, look in the error.log file in the logs subdirectory. If your host is not connected to the net, or if you have serious problems with your DNS (Domain Name Service) configuration, you may have to use this URL:

```
http://127.0.0.1/
```

If you happen to be running Apache on an alternate port, you need to explicitly put that in the URL:

```
http://127.0.0.1:8080/
```

Once your basic installation is working, you should configure it properly by editing the files in the conf subdirectory. Again, if you change the configuration of the Windows NT service for Apache, first attempt to start it from the command line to make sure that the service starts with no errors.

Because Apache **cannot** share the same port with another TCP/IP application, you may need to stop, uninstall or reconfigure certain other services before running Apache. These conflicting services include other WWW servers, some firewall implementations, and even some client applications (such as Skype) which will use port 80 to attempt to bypass firewall issues.

Configuring Access to Network Resources

Access to files over the network can be specified using two mechanisms provided by Windows:

Mapped drive letters
e.g., `Alias /images/ Z:/`

UNC paths
e.g., `Alias /images/ //imagehost/www/images/`

Mapped drive letters allow the administrator to maintain the mapping to a specific machine and path outside of the Apache httpd configuration. However, these mappings are associated only with interactive sessions and are not directly available to Apache httpd when it is started as a service. **Use only UNC paths for network resources in httpd.conf** so that the resources can be accessed consistently regardless of how Apache httpd is started. (Arcane and error prone procedures may work around the restriction on mapped drive letters, but this is not recommended.)

Example DocumentRoot with UNC path

```
DocumentRoot "//dochost/www/html/"
```

Example DocumentRoot with IP address in UNC path

```
DocumentRoot "//192.168.1.50/docs/"
```

Example Alias and corresponding Directory with UNC path

```
Alias "/images/" "//imagehost/www/images/"

<Directory "//imagehost/www/images/">
#...
</Directory>
```

When running Apache httpd as a service, you must create a separate account in order to access network resources, as described above.

Windows Tuning

- If more than a few dozen piped loggers are used on an operating system instance, scaling up the "desktop heap" is often necessary. For more detailed information, refer to the piped logging documentation.

Using Apache With Novell NetWare

This document explains how to install, configure and run Apache 2.0 under Novell NetWare 6.0 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 6.0 service pack 3 and above. If you are running a service pack less than SP3, you must install the latest NetWare Libraries for C (LibC).

NetWare service packs are available here.

Apache 2.0 for NetWare can also be run in a NetWare 5.1 environment as long as the latest service pack or the latest version of the NetWare Libraries for C (LibC) has been installed . **WARNING:** Apache 2.0 for NetWare has not been targeted for or tested in this environment.

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. Binary builds of the latest releases of Apache 2.0 for NetWare can be downloaded from here.

Installing Apache for NetWare

There is no Apache install program for NetWare currently. If you are building Apache 2.0 for NetWare from source, you will need to copy the files over to the server manually.

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` along with any file path values to reflect your correct server settings
- Add `SYS:/APACHE2` to the search path, for 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:/apache2`):

- Create a directory called `Apache2` on a NetWare volume
- Copy `APACHE2.NLM`, `APRLIB.NLM` to `SYS:/APACHE2`

- Create a directory under `SYS:/APACHE2` called `BIN`
- Copy `HTDIGEST.NLM`, `HTPASSWD.NLM`, `HTDBM.NLM`, `LOGRES.NLM`, `ROTLOGS.NLM` to `SYS:/APACHE2/BIN`
- 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`, `CHARSET.CONV` 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, for example:

```
SEARCH ADD SYS:\APACHE2
```

Apache may be installed to other volumes besides the default `SYS` volume.

During the build process, adding the keyword "install" to the makefile command line will automatically produce a complete distribution package under the subdirectory `DIST`. Install Apache by simply copying the distribution that was produced by the makfiles to the root of a NetWare volume (see: Compiling Apache for NetWare below).

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 apache2. 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. These directives are only available while Apache is running. 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, the state of each running threads is displayed on the Apache console screen.

SHUTDOWN
: Terminates the running instance of the Apache web server.

HELP
: Describes each of the runtime directives.

By default these directives are issued against the instance of Apache running in the OS address space. To issue a directive against a specific instance running in

a protected address space, include the -p parameter along with the name of the address space. For more information type "apache2 Help" on the command line.

Configuring Apache for NetWare

Apache is configured by reading configuration files usually stored 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 module 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 on some Unix implementations. Instead there are only threads running: a parent thread, and multiple child or worker threads which handle the requests.

  Therefore the "process"-management directives are different:

  `MaxConnectionsPerChild` - Like the Unix directive, this controls how many connections a worker thread will serve before exiting. The recommended default, `MaxConnectionsPerChild 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 must use NetWare filenames instead of Unix names. However, because Apache uses Unix-style names internally, forward slashes must be used rather than backslashes. It is recommended that all rooted file paths begin with a volume name. If omitted, Apache will assume the `SYS:` volume which may not be correct.

- 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:

  ```
  LoadModule status_module modules/status.nlm
  ```

  Information on creating loadable modules is also available.

Additional NetWare specific directives:

- CGIMapExtension - This directive maps a CGI file extension to a script interpreter.
- SecureListen - Enables SSL encryption for a specified port.
- NWSSLTrustedCerts - Adds trusted certificates that are used to create secure connections to proxied servers.
- NWSSLUpgradeable - Allow a connection created on the specified address/port to be upgraded to an SSL connection.

Compiling Apache for NetWare

Compiling Apache requires MetroWerks CodeWarrior 6.x or higher. Once Apache has been built, it can be installed to the root of any NetWare volume. 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. Alternatively, a complete distribution can be built by including the keyword `install` when invoking the makefiles.

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)
- LDAP Libraries for C
- ZLIB Compression Library source code
- 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 NetWare makefiles:

- Set the environment variable `NOVELLLIBC` to the location of the NetWare Libraries for C SDK, for example:

```
Set NOVELLLIBC=c:\novell\ndk\libc
```

- Set the environment variable `METROWERKS` to the location where you installed the Metrowerks CodeWarrior compiler, for example:

```
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 `LDAPSDK` to the location where you installed the LDAP Libraries for C, for example:

```
Set LDAPSDK=c:\Novell\NDK\cldapsdk\NetWare\libc
```

- Set the environment variable `ZLIBSDK` to the location where you installed the source code for the ZLib Library, for example:

```
Set ZLIBSDK=D:\NOVELL\zlib
```

- Set the environment variable `PCRESDK` to the location where you installed the source code for the PCRE Library, for example:

```
Set PCRESDK=D:\NOVELL\pcre
```

- Set the environment variable `AP_WORK` to the full path of the `httpd` source code directory.

```
Set AP_WORK=D:\httpd-2.0.x
```

- Set the environment variable `APR_WORK` to the full path of the `apr` source code directory. Typically `\httpd\srclib\apr` but the APR project can be outside of the httpd directory structure.

```
Set APR_WORK=D:\apr-1.x.x
```

- Set the environment variable `APU_WORK` to the full path of the `apr-util` source code directory. Typically `\httpd\srclib\apr-util` but the APR-UTIL project can be outside of the httpd directory structure.

```
Set APU_WORK=D:\apr-util-1.x.x
```

- 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-2.0` and build the prebuild utilities by running "`gmake -f nwgnumakefile prebuild`". This target will create the directory `\httpd-2.0\nwprebuild` and copy each of the utilities to this location that are necessary to complete the following build steps.
- Copy the files `\httpd-2.0\nwprebuild\GENCHARS.nlm` and `\httpd-2.0\nwprebuild\DFTABLES.nlm` 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-2.0\os\netware` on the build machine.
- Change directory to `\httpd-2.0` and build Apache by running "`gmake -f nwgnumakefile`". You can create a distribution directory by adding an install parameter to the command, for example:

```
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 prebuild`
  Builds all of the prebuild utilities and copies them to the `\nwprebuild` 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.o` or `\debug.o` 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.

Additional environment variable options

- To build all of the experimental modules, set the environment variable `EXPERIMENTAL`:

  ```
  Set EXPERIMENTAL=1
  ```

- To build Apache using standard BSD style sockets rather than Winsock, set the environment variable `USE_STDSOCKETS`:

  ```
  Set USE_STDSOCKETS=1
  ```

Building mod_ssl for the NetWare platform

By default Apache for NetWare uses the built-in module `mod_nw_ssl` to provide SSL services. This module simply enables the native SSL services implemented in NetWare OS to handle all encryption for a given port. Alternatively, mod_ssl can also be used in the same manner as on other platforms.

Before mod_ssl can be built for the NetWare platform, the OpenSSL libraries must be provided. This can be done through the following steps:

- Download the recent OpenSSL 0.9.8 release source code from the OpenSSL Source page (older 0.9.7 versions need to be patched and are therefore not recommended).
- Edit the file `NetWare/set_env.bat` and modify any tools and utilities paths so that they correspond to your build environment.
- From the root of the OpenSSL source directory, run the following scripts:

  ```
  Netware\set_env netware-libc
  Netware\build netware-libc
  ```

  For performance reasons you should enable to build with ASM code. Download NASM from the SF site. Then configure OpenSSL to use ASM code:

  ```
  Netware\build netware-libc nw-nasm enable-mdc2
  enable-md5
  ```

  Warning: don't use the CodeWarrior Assembler - it produces broken code!
- Before building Apache, set the environment variable `OSSLSDK` to the full path to the root of the openssl source code directory, and set WITH_MOD_SSL to 1.

  ```
  Set OSSLSDK=d:\openssl-0.9.8x
  Set WITH_MOD_SSL=1
  ```

站点导航

此翻译可能过期。要了解最近的更改，请阅读英文版。

本页列出了 Apache HTTP 服务器 2.5 的全部文档。

发行说明

- 从 2.2 升级到 2.4
- Apache 2.3/2.4 的新特性
- Apache 2.1/2.2 的新特性
- Apache 2.0 的新特性
- Apache 许可证

使用 Apache HTTP 服务器

- 编译与安装 Apache
- 启动 Apache
- 停止与重启 Apache
- 配置文件
- 配置片段
- 缓存指南
- 服务器全局配置
- 日志文件
- 从 URL 映射到文件系统
- 安全技巧
- 动态共享对象(DSO)
- 内容协商
- 定制错误响应
- 绑定指定地址与端口
- 多处理模块(MPM)
- 环境变量
- Apache 的处理器
- 过滤器
- 执行 CGI 前的用户切换(suEXEC)
- 性能调谐
- 常见问题

Apache 虚拟主机文档

- 概述

- 基于名称的虚拟主机
- 基于 IP 的虚拟主机
- 动态配置的大规模虚拟主机
- 虚拟主机样例
- 虚拟主机匹配的深入讨论
- 文件句柄限制
- Apache 的 DNS 相关问题

URL 改写指南

- 概述

- mod_rewrite 参考文档
- 简介
- 标志
- 技术细节

- 重新映射 URL
- 访问控制
- 高级技术

Apache SSL/TLS 加密

- 概述

- SSL/TLS 加密: 简介
- SSL/TLS 加密: 兼容性
- SSL/TLS 加密: 常见操作
- SSL/TLS 加密: 常见问题

指南与教程

- 概述

- 认证，授权与访问控制
- CGI 与动态内容
- 服务器端插入
- .htaccess 文件
- 用户私人网站目录(public_html)

平台相关说明

- 概述

- 在 Microsoft Windows 中使用 Apache
- 为 Microsoft Windows 编译 Apache
- 在 Novell NetWare 中使用 Apache
- 在 HPUX 中运行高性能 web 服务器

Apache HTTP 服务器与支持程序

- 概述

- 手册: httpd
- 手册: ab
- 手册: apachectl
- 手册: apxs
- 手册: configure
- 手册: dbmmanage
- 手册: htcacheclean
- 手册: htdbm
- 手册: htdigest
- 手册: htpasswd
- 手册: logresolve
- 手册: rotatelogs
- 手册: suexec
- 其它程序

Apache 杂项文档

- 概述

- 相关标准

Apache 模块

- 描述模块的术语
- 描述指令的术语

- Apache 核心特性
- Apache MPM 通用指令
- Apache MPM event
- Apache MPM netware
- Apache MPM os2
- Apache MPM prefork
- Apache MPM winnt
- Apache MPM worker

- Apache 模块 mod_access_compat
- Apache 模块 mod_actions
- Apache 模块 mod_alias
- Apache 模块 mod_allowhandlers
- Apache 模块 mod_allowmethods
- Apache 模块 mod_asis
- Apache 模块 mod_auth_basic
- Apache 模块 mod_auth_digest
- Apache 模块 mod_auth_form
- Apache 模块 mod_authn_anon
- Apache 模块 mod_authn_core
- Apache 模块 mod_authn_dbd
- Apache 模块 mod_authn_dbm
- Apache 模块 mod_authn_file
- Apache 模块 mod_authn_socache
- Apache 模块 mod_authnz_fcgi
- Apache 模块 mod_authnz_ldap
- Apache 模块 mod_authz_core
- Apache 模块 mod_authz_dbd
- Apache 模块 mod_authz_dbm
- Apache 模块 mod_authz_groupfile
- Apache 模块 mod_authz_host
- Apache 模块 mod_authz_owner
- Apache 模块 mod_authz_user
- Apache 模块 mod_autoindex
- Apache 模块 mod_brotli
- Apache 模块 mod_buffer
- Apache 模块 mod_cache
- Apache 模块 mod_cache_disk
- Apache 模块 mod_cache_socache
- Apache 模块 mod_cern_meta
- Apache 模块 mod_cgi
- Apache 模块 mod_cgid
- Apache 模块 mod_charset_lite
- Apache 模块 mod_crypto
- Apache 模块 mod_data
- Apache 模块 mod_dav
- Apache 模块 mod_dav_fs
- Apache 模块 mod_dav_lock
- Apache 模块 mod_dbd
- Apache 模块 mod_deflate
- Apache 模块 mod_dialup
- Apache 模块 mod_dir
- Apache 模块 mod_dumpio
- Apache 模块 mod_echo
- Apache 模块 mod_env
- Apache 模块 mod_example_hooks
- Apache 模块 mod_expires
- Apache 模块 mod_ext_filter
- Apache 模块 mod_file_cache
- Apache 模块 mod_filter
- Apache 模块 mod_firehose
- Apache 模块 mod_headers

- Apache 模块 mod_heartbeat
- Apache 模块 mod_heartmonitor
- Apache 模块 mod_http2
- Apache 模块 mod_ident
- Apache 模块 mod_imagemap
- Apache 模块 mod_include
- Apache 模块 mod_info
- Apache 模块 mod_isapi
- Apache 模块 mod_journald
- Apache 模块 mod_lbmethod_bybusyness
- Apache 模块 mod_lbmethod_byrequests
- Apache 模块 mod_lbmethod_bytraffic
- Apache 模块 mod_lbmethod_heartbeat
- Apache 模块 mod_ldap
- Apache 模块 mod_log_config
- Apache 模块 mod_log_debug
- Apache 模块 mod_log_forensic
- Apache 模块 mod_logio
- Apache 模块 mod_lua
- Apache 模块 mod_macro
- Apache 模块 mod_md
- Apache 模块 mod_mime
- Apache 模块 mod_mime_magic
- Apache 模块 mod_negotiation
- Apache 模块 mod_nw_ssl
- Apache 模块 mod_policy
- Apache 模块 mod_privileges
- Apache 模块 mod_proxy
- Apache 模块 mod_proxy_ajp
- Apache 模块 mod_proxy_balancer
- Apache 模块 mod_proxy_connect
- Apache 模块 mod_proxy_express
- Apache 模块 mod_proxy_fcgi
- Apache 模块 mod_proxy_fdpass
- Apache 模块 mod_proxy_ftp
- Apache 模块 mod_proxy_hcheck
- Apache 模块 mod_proxy_html
- Apache 模块 mod_proxy_http
- Apache 模块 mod_proxy_http2
- Apache 模块 mod_proxy_scgi
- Apache 模块 mod_proxy_uwsgi
- Apache 模块 mod_proxy_wstunnel
- Apache 模块 mod_ratelimit
- Apache 模块 mod_reflector
- Apache 模块 mod_remoteip
- Apache 模块 mod_reqtimeout
- Apache 模块 mod_request
- Apache 模块 mod_rewrite
- Apache 模块 mod_sed
- Apache 模块 mod_session
- Apache 模块 mod_session_cookie
- Apache 模块 mod_session_crypto
- Apache 模块 mod_session_dbd
- Apache 模块 mod_setenvif
- Apache 模块 mod_slotmem_plain
- Apache 模块 mod_slotmem_shm
- Apache 模块 mod_so
- Apache 模块 mod_socache_dbm
- Apache 模块 mod_socache_dc
- Apache 模块 mod_socache_memcache
- Apache 模块 mod_socache_redis
- Apache 模块 mod_socache_shmcb

- Apache 模块 mod_speling
- Apache 模块 mod_ssl
- Apache 模块 mod_ssl_ct
- Apache 模块 mod_status
- Apache 模块 mod_substitute
- Apache 模块 mod_suexec
- Apache 模块 mod_syslog
- Apache 模块 mod_systemd
- Apache 模块 mod_unique_id
- Apache 模块 mod_unixd
- Apache 模块 mod_userdir
- Apache 模块 mod_usertrack
- Apache 模块 mod_version
- Apache 模块 mod_vhost_alias
- Apache 模块 mod_watchdog
- Apache 模块 mod_xml2enc

开发者文档

- 概述

- Apache API 说明
- 在 APR 中调试内存分配
- Apache 2.x 文档
- Apache 2.x 钩子函数
- 将模块从 Apache 1.3 移植到 Apache 2.x
- Apache 2.x 中的请求处理
- Apache 2.x 中的过滤器

术语与索引

- 术语
- 模块索引
- 指令索引
- 指令快速参考

Apache 2.0 开发者文档

此翻译可能过期。要了解最近的更改，请阅读英文版。

开发者页面的许多文档都来自于 Apache 1.3。当更新到 Apache 2 时，它们可能位于不同的阶段。请耐心等待，或者直接向 dev@httpd.apache.org 邮件列表报告开发者页面的差异或错误。

主题

- Apache 1.3 API 说明
- 在 Apache 2.3/2.4 中的 API 改变
- Apache 2.x 钩子函数
- Apache 2.x 中的请求处理
- Apache 2.x 中的过滤器
- Apache 2.x 中的输出过滤器指南
- 将模块从 Apache 1.3 移植到 Apache 2.x
- 在 APR 中调试内存分配
- Apache 2.x 文档
- Apache 2.x 的线程安全问题

外部资源

- 自动生成的 Apache HTTP 服务器 (trunk) 代码文档
- Kevin O'Donnell 的模块开发教程
  - 集成模块到 Apache 构建系统
  - 处理配置指令

- Ryan Bloom 对 Apache 模块开发的说明
- 位于 apachetutor 的开发者文章:
  - Apache 中的请求处理
  - 模块的配置
  - Apache 中的资源管理
  - Apache 中的连接池
  - 桶与队列简介

Apache 杂项文档

此翻译可能过期。要了解最近的更改，请阅读英文版。

下面是适用于 Apache 服务器开发项目的附加文档。

警告

下面的文档尚未完全更新，以反映自 Apache HTTP 服务器版本 2.1 之后的修改。某些信息可能仍旧适用，但请小心使用它。

Apache 性能调谐

对如何在编译或运行时，配置 Apache，以便性能更高的说明。 解释了为什么 Apache 这样做，而不那样做 (这会让它更慢或更快)。

安全技巧

做和不做 - 如何让你的 Apache 站点保持安全。

相关标准

这篇文档是 Apache 遵循的相关标准的参考页面。

密码加密格式

对 Apache 身份认证支持的各种密码加密格式的讨论。