Apache HTTP 2.4
<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 2.3/2.4</td>
</tr>
<tr>
<td>Apache 2.1/2.2</td>
</tr>
<tr>
<td>Apache 2.0</td>
</tr>
<tr>
<td>2.2  2.4</td>
</tr>
<tr>
<td>Apache</td>
</tr>
</tbody>
</table>
(MPM)

*Expression parser*
Getting Started

(DSO)

URL

SSL/TLS

CGI  Suexec

mod_rewrite  URL Rewriting
How-To

CGI:

.htaccess

Server Side Includes (SSI)

(public_html)
Microsoft Windows
RPM (Redhat / CentOS / Fedora)
Novell NetWare
EBCDIC
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.

See also

Overview of new features in Apache HTTP Server 2.4
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
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](https://httpd.apache.org/docs/2.4/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: /var/log/access.log

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**

```html
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:**

```html
Order Deny,Allow
Deny from all
AuthBasicProvider File
AuthUserFile /example.com/conf/users.passwd
AuthName secure
Require valid-user
```
2.4 configuration:

```
# No replacement needed
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
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 either criteria:

2.4 configuration:

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

2.2 configuration:

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

### 2.4 configuration:

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 0n or 0ff. 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=12345678 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**: `DumpIOLLogLevel` 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](https://httpd.apache.org/docs/2.4/logs.html).
- **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.
- The RewriteLog and RewriteLogLevel directives have been removed. This functionality is now provided by configuring the appropriate level of logging for the mod_rewrite module using the LogLevel directive. See also the mod_rewrite logging section.
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.
• 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.
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.
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.
**mod_proxy_fcg**
FastCGI Protocol backend for mod_proxy

**mod_proxy_scg**
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_hcheck (available since 2.4.21)
  Support independent dynamic health checks for remote proxy backend servers.
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. Headers containing invalid characters (including underscores) are now silently dropped.

[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).
**fcgistarter**
New FastCGI daemon starter utility

**htcache_clean**
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).
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.
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.
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.
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.
**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.
**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](https://www.apache.org). 

**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.
Apache 2.0

Apache HTTP 1.3 2.0
Unix

POSIX Unix
autoconf  libtool
Apache
Apache

Apache

Unix

Apache 2.0  BeOSOS/2Windows  Unix
(MPM)  Apache Portable Runtime (APR)  API
POSIX

Apache API

2.0  API
1.3 /

IPv6

Apache  Apache Portable Runtime library  IPv6
Apache  IPv6 listen  Listen
NameVirtualHost, VirtualHost  IPv6
"Listen [2001:db8::1]:8080"

Apache
Server Side Include


SSI

vhost

Windows NT  Unicode
| Windows NT  | Apache 2.0                  | utf-8 |
| Windows 2000 | Windows XP               | Windows NT |
| Windows 95, 98, ME |

| Apache 2.0 | Perl (PCRE) | Perl |
mod_ssl
  Apache 2.0 OpenSSL
  SSL/TLS

mod_dav
  Apache 2.0
  Versioning (DAV)

mod_deflate
  Apache 2.0

mod_auth_ldap
  Apache 2.0.41 HTTP
  L

mod_auth_digest

mod_charset_lite
  Apache 2.0

mod_file_cache
  Apache 2.0
  Apache 1.3
  m

mod_headers
  Apache 2.0
  mod_pro;

mod_proxy
  proxy
  HTTP/1.1 pro
  proxy ()
  proxy_connect, proxy_ftp, proxy_http

mod_negotiation
  NOT ACCEPTABLE  MULTIPLE CHOICES
  ForceLanguagePriority

mod_autoindex
  Autoindex
  HTML
mod_include

SSI

mod_include $0 .. $9

mod_auth_dbm

AuthDBMType DBM
The Apache License, Version 2.0

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
Licensors 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.

```text
Copyright [yyyy] [name of copyright owner]

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

Copyright 2017 The Apache Software Foundation.
Licensed under the Apache License, Version 2.0.
Unix  Unix

Apache

Apache 2.0  Open Source

Apache  Windows

libtool

(2.2.50  2.2.51)
<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>$ lynx <a href="http://httpd.apache.org/download.cgi">http://httpd.apache.org/download.cgi</a></td>
</tr>
<tr>
<td>$ gzip -d httpd-NN.tar.gz</td>
</tr>
<tr>
<td>$ tar xvf httpd-NN.tar</td>
</tr>
<tr>
<td>$ cd httpd-NN</td>
</tr>
<tr>
<td>$ ./configure --prefix=PREFIX</td>
</tr>
<tr>
<td>$ make</td>
</tr>
<tr>
<td>$ make install</td>
</tr>
<tr>
<td>$ vi PREFIX/conf/httpd.conf</td>
</tr>
<tr>
<td>$ PREFIX/bin/apachectl -k start</td>
</tr>
</tbody>
</table>

```
NN 
PREFIX
/usr/local/apache2

Apache httpd
```
Apache:

50 MB

Apache 10!

ANSI-C

Free Software Foundation (FSF) GNU C

ANSI-C compiler (GCC) GCC

make

HTTP

xntpd NTP NTP

Perl 5 []

apxs dbmmanage Perl Perl 5 (5.003

configure

Apache httpd
Apache HTTP

Apache HTTP
Apache HTTPD tarball

```
$ gzip -d httpd-NN.tar.gz
$ tar xvf httpd-NN.tar
```
Apache
libtool buildconf

./configure

Apache --prefix /sw/pkg/apache

enable-module module
enable-module=shared (DSO)
disable-module Base

configure
configure --help

/sw/pkg/

DSO :

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

configure

Makefile

configure configure
Apache

$ make
$ make install

PREFIX
$ vi PREFIX/conf/httpd.conf

docs/manual/  Apache
  http://httpd.apache.org/docs/2.4/
Apache HTTP :

$ PREFIX/bin/apachectl -k start

URL http://localhost/
PREFIX/htdocs/ :

$ PREFIX/bin/apachectl -k stop
API

(2.2.55 2.2.57)

configure

config.nice

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

configure

config.nice:

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

Windows Apache

Unix   httpd

httpd
apachectl
Listen 80 (1024)

httpd
apachectl
HTTPD
httpd
httpd
httpd.conf

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

- root
- Apache

Apache FAQ
root

```
apachectl  SysV init
httpd  init
```

▲
Apache HTTP Server

Unix
Windows 9x, ME

Httpd
Apachectl
Starting
Apache HTTP Server

httpd
TERM, HUP, USR1, WINCH

```
kill -TERM `cat /usr/local/apache2/logs/httpd.pid`
```

```
httpd 2 -k graceful-stop
```

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

ServerRoot  PidFile
: TERM
   apachectl -k stop

TERM    stop kill
apachectl -k graceful

mod_status

status

()
HUP
    apachectl -k restart

HUP  restart
    TERM  kill

mod_status  HUP

graceful
Signal: WINCH
   apachectl -k graceful-stop

WINCH  graceful-stop
Listen
TERM

"graceful"  TERM

graceful-stop  httpd
(Mutex)  Unix

rotatelogs
Apache HTTP
<table>
<thead>
<tr>
<th>mod_mime</th>
<th><code>&lt;IfDefine&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Include</td>
<td>TypesConfig</td>
</tr>
</tbody>
</table>

Apache

Include

MIME
Apache 11

"\""

"#"

apachectl configtest
mod_so  <IfModule>
  LoadModule
</IfModule>

Apache

LoadModule

-l
<Directory>
  <DirectoryMatch>
  <Files>
  <FilesMatch>
  <Location>
  <LocationMatch>
  <VirtualHost>

  <FilesMatch>, <Location>, <LocationMatch>

  Apache
Apache
.htaccess
.htaccess
.htaccess
URL .htaccess
<Directory>
        <DirectoryMatch>
        <Files>
            <FilesMatch>
                <IfDefine>
                    <IfModule>
                        <IfVersion>
                            <Location>
                                <LocationMatch>
                                    <Proxy>
                                        <ProxyMatch>
                                            <VirtualHost>
                                            <IfDefine>, <IfModule>,
                                            <IfVersion>
                                            <IfDefine> httpd
                                            httpd -DClosedForNow :
                                            <IfDefine ClosedForNow>
                                            Redirect / http://otherserver.example.com/
                                            </IfDefine>
                                            <IfModule>
                                            LoadModule
                                            mod_mime_magic
                                            MimeMagicFiles
                                            <IfModule mod_mime_magic.c>
                                            MimeMagicFile conf/magic
                                            </IfModule>
                                            <VirtualHost>
<IfVersion>  <IfDefine>  <IfModule>  
httpd

<IfVersion >= 2.1>
  # this happens only in versions greater or
  # equal 2.1.0.
</IfVersion>

<IfDefine>, <IfModule>, <IfVersion>  "!"
Apache Unix
/usr/local/apache2

Windows
"c:/Program Files/Apache Group/Apache2" (Apache Windows)
/usr/local/apache2/htdocs/dir/

<Directory> <Files> <Directory>
.htaccess /var/web/dir1

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

<Files> private.html

<Files private.html>
Order allow,deny
Deny from all
</Files>

<Files> <Directory>
/var/web/dir1/private.html,
/var/web/dir1/subdir2/private.html,
/var/web/dir1/subdir3/private.html
/var/web/dir1/ private.html

<Directory /var/web/dir1>
<Files private.html>
Order allow,deny
Deny from all
</Files>
</Directory>
<Location> /private URL
http://yoursite.example.com/private,
http://yoursite.example.com/private123,
http://yoursite.example.com/private/dir/file.html
/private

<Location /private>
Order Allow,Deny
Deny from all
</Location>

<Location> URL mod_status Apache server-status

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

<Directory>, <Files>, <Location> C shell "** "?" 1 "[regex]

(regex) <DirectoryMatch>, <FilesMatch>,<LocationMatch> perl regex

regex

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

regex
<FilesMatch \.\.(?i:gif|jpe?g|png)$>
Order allow,deny
Deny from all
</FilesMatch>

<Directory>
<Files/>

<Location> (URL)

<Location /dir/>
Order allow,deny
Deny from all
</Location>

http://yoursite.example.com/dir/ ?
http://yoursite.example.com/DIR/
(Options)

<Location /> URL
<Proxy>
  <ProxyMatch> URL mod_proxy
cnn.com

<Proxy http://cnn.com/*>
Order allow,deny
Deny from all
</Proxy>
Context  <Directory>  <DirectoryMatch>,
<Files>, <FilesMatch>, <Location>, <LocationMatch>,
<Proxy>, <ProxyMatch>

- AllowOverride <Directory>
- FollowSymLinks  SymLinksIfOwnerMatch  Options
  <Directory>  .htaccess
- Options  <Files>  <FilesMatch>
1. `<Directory> () .htaccess ( .htaccess <Directory> )`

2. `<DirectoryMatch> ( <Directory ~>
   <Directory>`

3. `<Files>  <FilesMatch>`

4. `<Location>  <LocationMatch>`

   `<Directory>`
   `<Directory /var/web/dir1>  `<Directory /var/web/dir/subdir>`
   Include Include

   `<VirtualHost`

   mod_proxy `<Directory>`

   (URL Alias
   `<Location> </LocationMatch>`

   A > B > C > D > E

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

   `<Files f.html>`
   D`
   `</Files>`
<VirtualHost *>
<Directory /a/b>
  B
</Directory>
</VirtualHost>

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

<Directory /a/b>
  A
</Directory>

<Directory>
  <Location>
    Order deny,allow
    Allow from all
  </Location>

  # Woops! This <Directory> section will have no effect
  <Directory />
  Order allow,deny
  Allow from all
  Deny from badguy.example.com
</Directory>
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.
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.
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](https://www.rfc-editor.org/rfc/rfc2616#section-13.4), and can be summed up as follows:

1. Caching must be enabled for this URL. See the `CacheEnable` and `CacheDisable` directives.

2. The response must have a HTTP status code of 200, 203, 300, 301 or 410.

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 uncachable 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.
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 xyTGxSM02b68mBCykqkp1w. 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/TGxSM02b68mBCykqkp1w.

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.
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:112:
```
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:1121,
```

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
```
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

<table>
<thead>
<tr>
<th>Related Modules</th>
<th>Related Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>mod_authn_socache</code></td>
<td><code>AuthnCacheSOCache</code></td>
</tr>
</tbody>
</table>

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

## Caching SSL Sessions

<table>
<thead>
<tr>
<th>Related Modules</th>
<th>Related Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_ssl</td>
<td>SSLSessionCache</td>
</tr>
<tr>
<td></td>
<td>SSLStaplingCache</td>
</tr>
</tbody>
</table>

The *mod_ssl* module uses the socache interface to provide a session cache and a stapling cache.
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.
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.
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.
core
ServerName
ServerAdmin
ServerSignature
ServerTokens
UseCanonicalName
UseCanonicalPhysicalPort

ServerAdmin  ServerTokens
Server HTTP

ServerName, UseCanonicalName, UseCanonicalPhysicalPort  URL
Apache
CoreDumpDirectory
DocumentRoot
ErrorLog
LockFile
PidFile
ScoreBoardFile
ServerRoot

Apache
LimitRequestBody
LimitRequestFields
LimitRequestFieldsize
LimitRequestLine
RLimitCPU
RLimitMEM
RLimitNPROC
ThreadStackSize

LimitRequest* Apache

RLimit* Apache  fork

ThreadStackSize Netware
ErrorLog
LogLevel

ErrorLog

(Unix
Unix
error_log
syslog

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

CGI

tail -f error_log
Apache httpd  mod_log_config, mod_log_agent
TransferLog

C  printf(1)

Common Log Format

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

Common Log Format (CLF)


127.0.0.1 (%h)
### Combined Log Format

**LogFormat** "%h %l %u %t "%r" %>s %b " %{Referer}i" " %{User-agent}i" combined

**CustomLog** log/access_log combined

### Common Log Format

**HTTP**

```
"http://www.example.com/start.html" "Mozilla/4.08 [en] (Win98; I ;Nav)"
```

```
"http://www.example.com/start.html"("%{Referer}i")
   "Referer"() HTTP
)
"Mozilla/4.08 [en] (Win98; I ;Nav)"("%{User-agent}i")
   User-Agent HTTP
```

**CustomLog** logs/access_log common
**CustomLog** logs/referer_log " %{Referer}i -> %U"
**CustomLog** logs/agent_log " %{User-agent}i"

**LogFormat**"%h %l %u %t "%r" %>s %b" common
# Mark requests from the loop-back interface
SetEnvIf Remote.Addr "127\..\..\..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

SetEnvIf Accept-Language "en" english
CustomLog logs/english_log common env=english
CustomLog logs/non_english_log common env=!english
10,000

open

open

open

mv access_log access_log.old
mv error_log error_log.old
apachectl graceful
sleep 600
gzip access_log.old error_log.old
<table>
<thead>
<tr>
<th>Apache httpd ID</th>
</tr>
</thead>
</table>

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

cronolog
CustomLog  ErrorLog  <VirtualHost>

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

%v
<table>
<thead>
<tr>
<th>Module</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_logio</td>
<td>LogFormat</td>
</tr>
<tr>
<td>mod_log_forensic</td>
<td>ForensicLog</td>
</tr>
<tr>
<td>mod_cgi</td>
<td>PidFile</td>
</tr>
<tr>
<td>mod_rewrite</td>
<td>RewriteLog</td>
</tr>
<tr>
<td></td>
<td>RewriteLogLevel</td>
</tr>
<tr>
<td></td>
<td>ScriptLog</td>
</tr>
<tr>
<td></td>
<td>ScriptLogBuffer</td>
</tr>
<tr>
<td></td>
<td>ScriptLogLength</td>
</tr>
</tbody>
</table>

**mod_logio** (%I %O)

**Forensic**

**mod_log_forensic** forensic

**PID**

Apache httpd ID logs/httpd.pid

PidFile ID

ScriptLog CGI

**mod_rewrite**

RewriteLogLevel
Apache License, Version 2.0.
URL

Apache URL
<table>
<thead>
<tr>
<th>mod_alias</th>
<th>Alias</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_proxy</td>
<td>AliasMatch</td>
</tr>
<tr>
<td>mod_rewrite</td>
<td>CheckSpelling</td>
</tr>
<tr>
<td>mod_userdir</td>
<td>DocumentRoot</td>
</tr>
<tr>
<td>mod_speling</td>
<td>ErrorDocument</td>
</tr>
<tr>
<td>mod_vhost_alias</td>
<td>Options</td>
</tr>
<tr>
<td></td>
<td>ProxyPass</td>
</tr>
<tr>
<td></td>
<td>ProxyPassReverse</td>
</tr>
<tr>
<td></td>
<td>ProxyPassReverseCookieDomain</td>
</tr>
<tr>
<td></td>
<td>ProxyPassReverseCookiePath</td>
</tr>
<tr>
<td></td>
<td>Redirect</td>
</tr>
<tr>
<td></td>
<td>RedirectMatch</td>
</tr>
<tr>
<td></td>
<td>RewriteCond</td>
</tr>
<tr>
<td></td>
<td>RewriteMatch</td>
</tr>
<tr>
<td></td>
<td>ScriptAlias</td>
</tr>
<tr>
<td></td>
<td>ScriptAliasMatch</td>
</tr>
<tr>
<td></td>
<td>UserDir</td>
</tr>
</tbody>
</table>
DocumentRoot

DocumentRoot

SymLinksIfOwnerMatch

Alias

Alias /docs /var/web


ScriptAlias CGI

AliasMatch ScriptAliasMatch

ScriptAliasMatch ^/-(?i:~[a-zA-Z0-9]+)/cgi-bin/(.+)$ /home/$1/cgi-bin/$2

http://example.com/~user/cgi-bin/script.cgi /home/user/cgi-bin/script.cgi CGI
Unix  user  ~user/  mod_userdir

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

URL  /home/user/public_html/file.html
     /home/user/  /etc/passwd

Userdir  /etc/passwd

"~"( %7e)

http://www.example.com/upages/user/file.html
/home/user/public_html/file.html

AliasMatch  ^/upages/([a-zA-Z0-9]+)/?(.*)
/home/$1/public_html/$2
Apache

DocumentRoot /foo/ /bar/

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

/foo/ URL-Path www.example.com /bar/
/foo/

Apache

RedirectMatch

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

RedirectMatch temp .*
http://othersite.example.com/startpage.html
Apache URL

/foo/ internal.example.com /bar/

ProxyPass /foo/ http://internal.example.com/bar/
ProxyPassReverse /foo/ http://internal.example.com/bar/
ProxyPassReverseCookieDomain internal.example.com
ProxyPassReverseCookieDomain public.example.com
ProxyPassReverseCookiePath /foo/ /bar/

ProxyPass
ProxyPassReverse
internal.example.com
ProxyPassReverseCookieDomain
ProxyPassReverseCookiePath
Cookie

internal.example.com
internal.example.com mod_proxy_html
HTML XHTML
mod_rewrite

() mod_rewrite
URL

"File Not Found"

mod_speling ()

Found"

mod_speling
mod_speling

ErrorDocument

Apache HTTP 404
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.
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.
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/](http://modules.apache.org/) that can restrict certain client behaviors and thereby mitigate DoS problems.
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 (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.
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.
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.
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.
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.
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.
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.
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:

```
/.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>
```
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.
Apache HTTP
Shared Object) (DSO)  

DSO
mod_so

Apache DSO

core.c DSO

enable-module=shared DSO

Apache DSO

Apache C DSO

httpd.conf

mod_so

Apache DSO

mod_so.c

mod_so

Apache DSO

Apache C DSO

Apache
Apache 2.x DSO:

1. Apache mod_foo.c DSO mod_foo.so:

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

2. Apache mod_foo.c DSO mod_foo.so:

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

3. Apache:

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

4. Apache mod_foo.c apxs Apache:

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

httpd.conf  Loa
Unix OS (DSO) /

: 1

DSO DSO DSO libfoo.so 1:
( /usr/lib)
/usr/lib -R
libfoo.so () DSO

DSO (DSO )
)

DSO DSO ( dlopen() DSO
dll)
)
)

DSO API

DSO : DSO

DSO

1998 DSO : Perl 5

Apache
Apache DSO
DSO:

- configure Apache (SSL [mod_perl, PHP3] Apache)
- Apache DSO apxs Apache
  apxs -i apachectl restart

DSO:

- Unix 20%
  (PIC) (position independent code)

- DSO DSO
  ) DSO DS
  Apache PI
dlopen ()
Apache HTTP/1.1

mod_negotiation
Accept-Language: fr

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

Apache HTTP/1.1 'server driven'
Language, Accept-Charset, Accept-Encoding Apache 'transparent'
'feature negotiation'

URI (RFC 2396) Apache HTTP 0 1
Apache

variant

- (*.var) variant
- 'Multiviews'

**type-map**

type-map (Apache MIME type-map)

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

variant

)

foo

foo.

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

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

**MultiViews**

"qs"

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
qs 0.000 1.000 qs
variant qs 1.0 qs
JPEG
qs variant

mod_negotiation

**Multiviews**

```
<Files>
  AllowOverride All
</Files>

MultiViews: /some/dir/foo
MultiViews /some/dir/foo
```

```
DirectoryIndex index
```

```
index.html index.html3
```

Multiviews
Apache variant

1. **Apache Server** driven negotiation
   - Apache

2. RFC 2295 transparent content negotiation
   - RFC 2296 'remote variant selection algorithm'

```plaintext
Accept
Accept-Language
Accept-Encoding
Accept-Charset
```

**Apache**

variant ()

1. `Accept*`

2. variant var

   1. variant Accept variant
   2. variant

3. () Accept-Language ()

   `LanguagePriority` variant
4. (text/html )

5. Accept-Charset
   text/*

6. ISO-8859-1
   user-agent

7. variant
   variant

8. variant

9. variant

3. variant

4. variant
   representation" )
   HTML

Vary
Apache
Accept

Accept:

Accept: image/*, */*

"image/"

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

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

1.0 ()

Accept: q
"type/*" 0.02 q
Apache */* 0.01 q
("*/*")

Apache 2.0

"Multiple Choices"

LanguagePriority
Language en-GB en
Acceptable Variants" LanguagePriority
en Apache "fr" "fr"

(Cookie URL)
mod_negotiation prefer-language
mod_negotiation variant

Example
SetEnvIf Cookie "language=(.+)") prefer-language=$1
Header append Vary cookie
Apache transparent content negotiation (RFC 2295) variant
{encoding ..} variant variant
Accept-Encoding variant variant RVSA/1.0
(RFC 2296) RVSA/1.0 variant 5
MIME ( html) ( gz)

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

<table>
<thead>
<tr>
<th>foo.html.en</th>
<th>foo</th>
<th>foo.html</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>foo.en.html</td>
<td>foo</td>
<td>foo.html</td>
<td>foo.html</td>
</tr>
<tr>
<td>foo.html.en.gz</td>
<td>foo</td>
<td>foo.html</td>
<td>foo.gz foo.html.gz</td>
</tr>
<tr>
<td>foo.en.html.gz</td>
<td>foo</td>
<td>foo.html</td>
<td>foo.html foo.html.gz foo.gz</td>
</tr>
<tr>
<td>foo.gz.html.en</td>
<td>foo</td>
<td>foo.gz</td>
<td>foo.html</td>
</tr>
<tr>
<td>foo.html.gz.en</td>
<td>foo</td>
<td>foo.html</td>
<td>foo.gz</td>
</tr>
</tbody>
</table>

( foo)

MIME ( foo.html) ()
URL

HTTP/1.0
HTTP/1.1

HTTP/1.0
HTTP/1.1
<table>
<thead>
<tr>
<th>Apache</th>
<th>HTTP</th>
<th>2.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache HTTP 2.4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Apache

"500 Server Error"
NCSA httpd 1.3 /

1. NCSA
2. URL
3. URL

URL /

Apache CGI:

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

REDIRECT_URL REDIRECT_QUERY_STRING URL (CGI CGI)
AllowOverride .htaccess ErrorDocument

ErrorDocument 500 /cgi-bin/crash-recover
ErrorDocument 500 "Sorry, our script crashed. Oh dear"
ErrorDocument 500 http://xxx/
ErrorDocument 404 /Lame_excuses/not_found.html
ErrorDocument 401 /Subscription/how_to_subscribe.html

ErrorDocument <3-digit-code> <action>

action ()

1. ("")
2. URL
3. URL
... print "Content-type: text/html\n";
printf "Status: %s Condition Intercepted\n",
$ENV{"REDIRECT_STATUS"};
...

404 Not Found
Apache

DNS
Apache
Apache

**Listen**

<table>
<thead>
<tr>
<th>Listen</th>
<th>Listen</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>8000</td>
</tr>
<tr>
<td>Listen 80</td>
<td>Listen 8000</td>
</tr>
<tr>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>Listen 192.0.2.1:80</td>
<td>Listen 192.0.2.5:8000</td>
</tr>
<tr>
<td>IPv6</td>
<td></td>
</tr>
<tr>
<td>Listen [2001:db8::a00:20ff:fea7:ccea]:80</td>
<td></td>
</tr>
<tr>
<td>APR</td>
<td>IPv6</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
</tr>
<tr>
<td>IPv6</td>
<td>IPv4</td>
</tr>
<tr>
<td>--enable-v4-mapped</td>
<td>configure</td>
</tr>
</tbody>
</table>

**Linux** *Tru64* **IPv4** **IPv6**

**IPv6**

**--enable-v4-mapped** **configure**

**FreeBSD, NetBSD, OpenBSD**

**Apache**

**APR** **IPv4**

**Listen 0.0.0.0:80**

**Listen 192.0.2.1:80**

**Apache** **IPv4** **IPv6**

**v4-mapped** **configure**

**mapped** **FreeBSD, NetBSD, OpenBSD**
Apache HTTP
Apache HTTP

Apache 2.0

- Apache
- Apache 1.3 POSIX

MPM Apache MPM
configure --with-mpm=NAME

./httpd -l
<table>
<thead>
<tr>
<th>OS</th>
<th>MPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>BeOS</td>
<td>beos</td>
</tr>
<tr>
<td>Netware</td>
<td>mpm_netware</td>
</tr>
<tr>
<td>OS/2</td>
<td>mpmt_os2</td>
</tr>
<tr>
<td>Unix</td>
<td>prefork</td>
</tr>
<tr>
<td>Windows</td>
<td>mpm_winnt</td>
</tr>
</tbody>
</table>
Apache

Apache HTTP
<table>
<thead>
<tr>
<th>Apache</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_setenvif</td>
</tr>
<tr>
<td>referrer HTTP</td>
</tr>
<tr>
<td>Referer</td>
</tr>
<tr>
<td>RewriteRule</td>
</tr>
<tr>
<td>[E=...]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>mod_unique_id</th>
</tr>
</thead>
</table>

**CGI**

- CGI
- CGI  `suexec` CGI
- `SetEnv`
<table>
<thead>
<tr>
<th>Module</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_authz_host</td>
<td>Allow</td>
</tr>
<tr>
<td>mod_cgi</td>
<td>CustomLog</td>
</tr>
<tr>
<td>mod_ext_filter</td>
<td>Deny</td>
</tr>
<tr>
<td>mod_headers</td>
<td>ExtFilterDefine</td>
</tr>
<tr>
<td>mod_include</td>
<td>Header</td>
</tr>
<tr>
<td>mod_log_config</td>
<td>LogFormat</td>
</tr>
<tr>
<td>mod_rewrite</td>
<td>RewriteCond</td>
</tr>
<tr>
<td></td>
<td>RewriteRule</td>
</tr>
</tbody>
</table>

### CGI

- **CGI**

### SSI

- **mod_include**  INCLUDES
- **server-parsed (SSI)**

**Allow from env** = **deny from env** =

**LogFormat**

- `%e
- `gif`

**Header**
ExtFilterDefine
enable env=

URL
RewriteCond %{ENV:...} mod_rewrite
ENV:
PassEnv

downgrade-1.0
HTTP/1.0 HTTP/1.0

force-gzip
DEFLATE accept-encoding

force-no-vary
Vary

force-response-1.0
HTTP/1.0 HTTP/1.1

gzip-only-text/html
1 text/html mod_deflate (gzip "identity")

no-gzip
  mod_deflate DEFLATE mod_neg

nokeepalive
  KeepAlive

prefer-language
  mod_negotiation (en, ja, x-klingon)

variant
redirect-carefully

suppress-error-charset

Apache 2.2

UTF-7

force-proxy-request-1.0, proxy-nokeepalive, proxy-sendchunked, proxy-sendcl

mod_proxy  mod_proxy
# The following directives modify normal HTTP response behavior.
# The first directive disables keepalive for Netscape 2.x and browsers that
# spoof it. There are known problems with these browser implementations.
# The second directive is for Microsoft Internet Explorer 4.0b2
# which has a broken HTTP/1.1 implementation and does not properly
# support keepalive when it is used on 301 or 302 (redirect) responses.
#
BrowserMatch "Mozilla/2" nokeepalive
BrowserMatch "MSIE 4\0b2;" nokeepalive downgrade-1.0 force-response-1.0

# The following directive disables HTTP/1.1 responses to browsers
# are in violation of the HTTP/1.0 spec by not being able to grok
# basic 1.1 response.
#
BrowserMatch "RealPlayer 4\0" force-response-1.0
BrowserMatch "Java/1\0.0" force-response-1.0
BrowserMatch "JDK/1\0.0" force-response-1.0

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

SetEnvIf Referer "^http://www\.example\.com/" local_referal
# Allow browsers that do not send Referer info
SetEnvIf Referer "^$" local_referal
<Directory /web/images>
    Order Deny,Allow
    Deny from all
    Allow from env=local_referal
</Directory>
<table>
<thead>
<tr>
<th>mod_actions</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_asis</td>
<td>AddHandler</td>
</tr>
<tr>
<td>mod_cgi</td>
<td>RemoveHandler</td>
</tr>
<tr>
<td>mod_imagemap</td>
<td>SetHandler</td>
</tr>
<tr>
<td>mod_info</td>
<td></td>
</tr>
<tr>
<td>mod_mime</td>
<td></td>
</tr>
<tr>
<td>mod_negotiation</td>
<td></td>
</tr>
<tr>
<td>mod_status</td>
<td></td>
</tr>
</tbody>
</table>

Apache

- **default-handler**: `default_handler()`
- **send-as-is**: HTTP
- **cgi-script**: CGI
- **imap-file**: (mod_imagemap)
- **server-info**: (mod_info)
- **server-status**: (mod_status)
- **type-map**:  


CGI

html footer.pl CGI

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

HTTP

send-as-is HTTP
send-as-is

<Directory /web/htdocs/asis>
    SetHandler send-as-is
</Directory>
char *handler

(: "-"
)
Apache HTTP 2.4
Apache
<table>
<thead>
<tr>
<th>mod_deflate</th>
<th>AddInputFilter</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_ext_filter</td>
<td>AddOutputFilter</td>
</tr>
<tr>
<td>mod_include</td>
<td>RemoveInputFilter</td>
</tr>
<tr>
<td></td>
<td>RemoveOutputFilter</td>
</tr>
<tr>
<td></td>
<td>ExtFilterDefine</td>
</tr>
<tr>
<td></td>
<td>ExtFilterOptions</td>
</tr>
<tr>
<td></td>
<td>SetInputFilter</td>
</tr>
<tr>
<td></td>
<td>SetOutputFilter</td>
</tr>
</tbody>
</table>

Apache HTTP

**INCLUDES**

- **mod_include**  Server-Side Include

**DEFLATE**

- **mod_deflate**

- **mod_ext_filter**
<table>
<thead>
<tr>
<th>suEXEC</th>
<th>Apache</th>
<th>Web</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSI</td>
<td>web</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CGI</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

suEXEC
Apache

1    setuid  setgid  UNIX

2

3    suEXEC  suEXEC
    Apache  suEXEC

4    suEXEC  Apache
    suEXEC  suEXEC
    suEXEC

? !
suEXEC

suEXEC Apache web  setuid "wrapper"
HTTP wrapper  Apache
wrapper

1. wrapper ?

wrapper

2. wrapper ?

wrapper  Apache
suEXEC

3. wrapper ?

wrapper ?  (Apache)

4. CGI, SSI ?

CGI, SSI '/'
'-.'?
-with-suexec-docroot=DIR)

5. ?

?

6. ?

?

7. ?

suEXEC  root CGI/SSI
8. **ID** **ID**

   ID

   CGI/SSI  ID

9. ?

   suEXEC 'root' CGI/SSI

10. **ID** **ID**

    ID

    CGI/SSI

11. **wraper**?

    setuid  setgid

12. CGI/SSI

    (change directory)

13. Apache?

    suEXEC?

    (suEXEC)

14. ?

15. CGI/SSI?

16. CGI/SSI

   CGI/SSI

17. CGI/SSI  setuid  setgid

   ?
UID/GID

18. /?

19. ?

   suEXEC

20. CGI/SSI exec?

   suEXEC

   suEXEC wrapper

   suEXEC
suEXEC

---enable-suexec

  enable-suexec

---with-suexec-bin=PATH

  suexec
  bin=/usr/sbin/suexec

---with-suexec-caller=UID

  Apache

---with-suexec-userdir=DIR

  suEXEC
  ("*")

  UserDir

---with-suexec-docroot=DIR

  Apache
  "htdocs"

  wrapper "/home/apache/htdocs"

---with-suexec-uidmin=UID

  suEXEC
  UID
  500 10

---with-suexec-gidmin=GID

  suEXEC
  GID
  100

---with-suexec-logfile=FILE

  suEXEC
  logfiledir

---with-suexec-safepath=PATH

  CGI
  "/usr/local/bin:/t"
suEXEC wrapper
--enable-suexec suEXEC
(Apache)
"make"
made install
"/usr/local/apache/bin/suexec"

root wrapper ID

suEXEC
--with-suexec-caller configure
suEXEC

User www
Group webgroup

suexec "/usr/local/apache2/bin/suexec"

chgrp webgroup /usr/local/apache2/bin/suexec
chmod 4750 /usr/local/apache2/bin/suexec

Apache
suEXEC
suEXEC
Apache --sbindir suexec
"/usr/local/apache/sbin/suexec"

suEXEC

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

wrapper

suEXEC Apache

suEXEC suexec

Apache kill

●
CGI suEXEC

::
suEXEC wrapper

ID CGI

<VirtualHost>

mod_userdir

--with-suexec-userdir
suEXEC wrapper       --with-suexec-logfile
Apache wrapper

- suEXEC

- suEXEC

- suEXEC

- suEXEC PATH

Copyright 2017 The Apache Software Foundation.
Apache License, Version 2.0.
Apache Performance Tuning

Apache 2.x 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.x is capable of high performance in many real-world situations.

Compared to Apache 1.3, release 2.x contains many additional optimizations to increase throughput and scalability. Most of these improvements are enabled by default. However, there are compile-time and run-time configuration choices that can significantly affect performance. This document describes the options that a server administrator can configure to tune the performance of an Apache 2.x 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.
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 2 to deliver static content faster and with lower CPU utilization.
### HostnameLookups and other DNS considerations

Prior to Apache 1.3, **HostnameLookups** defaulted to On. This adds latency to every request because it requires a DNS lookup to complete before the request is finished. In Apache 1.3 this setting defaults to Off. If you need to have addresses in your log files resolved to hostnames, use the `logresolve` program that comes with Apache, or one of the numerous log reporting packages which are available.

It is recommended that you do this sort of postprocessing 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, therefore, use IP

---

<table>
<thead>
<tr>
<th>Related Modules</th>
<th>Related Directives</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_dir</td>
<td>AllowOverride</td>
</tr>
<tr>
<td>mpm_common</td>
<td>DirectoryIndex</td>
</tr>
<tr>
<td>mod_status</td>
<td>HostnameLookups</td>
</tr>
<tr>
<td></td>
<td>EnableMMAP</td>
</tr>
<tr>
<td></td>
<td>EnableSendfile</td>
</tr>
<tr>
<td></td>
<td>KeepAliveTimeout</td>
</tr>
<tr>
<td></td>
<td>MaxSpareServers</td>
</tr>
<tr>
<td></td>
<td>MinSpareServers</td>
</tr>
<tr>
<td></td>
<td>Options</td>
</tr>
<tr>
<td></td>
<td>StartServers</td>
</tr>
</tbody>
</table>
addresses, rather than names, when using these directives, if possible.

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

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

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

**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:

```html
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,

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

and a request is made for the URI `/index.html`. Then Apache will attempt to open `/../.htaccess, /www/../.htaccess, and /www/htdocs/../.htaccess. The solutions are similar to the
previous case of Options FollowSymLinks. For highest performance use AllowOverride None everywhere in your filesystem.

**Negotiation**

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

```
DirectoryIndex index
```

Use a complete list of options:

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

where you list the most common choice first.

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

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.)

**Process Creation**

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

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

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

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

When keep-alives are in use, children will be kept busy doing nothing waiting for more requests on the already open connection. The default **KeepAliveTimeout** of 5 seconds attempts to minimize this effect. The tradeoff here is between network bandwidth and server resources. In no event should you raise this above about 60 seconds, as **most of the benefits are lost**.
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):

```c
for (;;) {
    for (;;) {
        fd_set accept_fds;

        FD_ZERO (&accept_fds);
        for (i = first_socket; i <= last_socket; ++i)
            FD_SET (i, &accept_fds);
    }
    rc = select (last_socket+1, &accept_fds, NULL, NULL, NULL);
    if (rc < 1) continue;
    new_connection = -1;
    for (i = first_socket; i <= last_socket; ++i)
        if (FD_ISSET (i, &accept_fds) &&
            new_connection = accept (i, NULL, NULL);
        if (new_connection != -1) b
```
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):

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

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

```c
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
        if (error) break;
        if (s is ready for reading) {
            if (read (s, junk_buffer, siz
                break;
        }
        /* just toss away whatever is
        }

    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.
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...)  = 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.

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.

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.

The worker thread reads the request from the client.
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.

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).

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.

The worker thread does a lingering close of the connection.

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.
<table>
<thead>
<tr>
<th>core</th>
<th>DocumentRoot</th>
<th>NameVirtualHost</th>
<th>ServerAlias</th>
<th>ServerName</th>
<th>ServerPath</th>
<th>VirtualHost</th>
</tr>
</thead>
</table>

```xml
NameVirtualHost * IP
* :80

<VirtualHost>
NameVirtualHost (IP
ServerName
</VirtualHost>

ServerName DocumentRoot

www.domain.tld

httpd.conf

NameVirtualHost * :80

<VirtualHost * :80>
 ServerName www.domain.tld
 ServerAlias domain.tld * .domain.tld
 DocumentRoot /www/domain
 </VirtualHost>

<VirtualHost * :80>
ServerName www.otherdomain.tld
DocumentRoot /www/otherdomain
</VirtualHost>

NameVirtualHost  VirtualHost  *
                   IP
                   IP

<VirtualHost>
  ServerAlias domain.tld  *.domain.tld

  domain.tld  www.domain.tld
  ServerName  ServerAlias

  <VirtualHost>
    <<<VirtualHost>>>  

    <NameVirtualHost>  IP
    <VirtualHost>  
      ServerName  ServerAlias
      IP
      IP

    NameVirtualHost  
    <VirtualHost>
ServerPath

NameVirtualHost 111.22.33.44

<VirtualHost 111.22.33.44>
  ServerName www.domain.tld
  ServerPath /domain
  DocumentRoot /web/domain
</VirtualHost>

?                      
"/domain" URL
http://www.domain.tld/domain/ Host
http://www.domain.tld/

http://www.domain.tld/domain/misc/file.html"../icons/image.gif")/domain/
(: "http://www.domain.tld/domain/misc/file.html" "/domain/misc/file.html")
Apache IP
Apache:

1 2

- IP listen
- httpd

httpd: listen

httpd listen }
httpd

Listen	www.smallco.com:80

IP
httpd

ServerName, DocumentRoot, ErrorLog, TransferLog
CustomLog

<VirtualHost www.smallco.com>
ServerAdmin webmaster@mail.smallco.com
DocumentRoot /groups/smallco/www
ServerName www.smallco.com
ErrorLog /groups/smallco/logs/error_log
TransferLog /groups/smallco/logs/access_log
</VirtualHost>

<VirtualHost www.baygroup.org>
ServerAdmin webmaster@mail.baygroup.org
DocumentRoot /groups/baygroup/www
ServerName www.baygroup.org
ErrorLog /groups/baygroup/logs/error_log
TransferLog /groups/baygroup/logs/access_log
</VirtualHost>

suEXEC

SuexecUserGroup

VirtualHost

: Apache
Dynamically Configured Mass Virtual Hosting

This document describes how to efficiently serve an arbitrary number of virtual hosts with the Apache HTTP Server. A separate document discusses using mod_rewrite to create dynamic mass virtual hosts.
The techniques described here are of interest if your httpd.conf contains many <VirtualHost> sections that are substantially the same, for example:

```
<VirtualHost 111.22.33.44>
  ServerName customer-1.example.com
  DocumentRoot /www/hosts/customer-1.example.com/docs
  ScriptAlias "/cgi-bin/" /www/hosts/customer-1.example.com/cgi-bin
</VirtualHost>

<VirtualHost 111.22.33.44>
  ServerName customer-2.example.com
  DocumentRoot /www/hosts/customer-2.example.com/docs
  ScriptAlias "/cgi-bin/" /www/hosts/customer-2.example.com/cgi-bin
</VirtualHost>

<VirtualHost 111.22.33.44>
  ServerName customer-N.example.com
  DocumentRoot /www/hosts/customer-N.example.com/docs
  ScriptAlias "/cgi-bin/" /www/hosts/customer-N.example.com/cgi-bin
</VirtualHost>
```

We wish to replace these multiple <VirtualHost> blocks with a mechanism that works them out dynamically. This has a number of advantages:

1. Your configuration file is smaller, so Apache starts more quickly and uses less memory. Perhaps more importantly, the smaller configuration is easier to maintain, and leaves less room for errors.

2. Adding virtual hosts is simply a matter of creating the appropriate directories in the filesystem and entries in the DNS - you don't need to reconfigure or restart Apache.
The main disadvantage is that you cannot have a different log file for each virtual host; however, if you have many virtual hosts, doing this can be a bad idea anyway, because of the number of file descriptors needed. It is better to log to a pipe or a fifo, and arrange for the process at the other end to split up the log files into one per virtual host. One example of such a process can be found in the split-logfile utility.
A virtual host is defined by two pieces of information: its IP address, and the contents of the Host: header in the HTTP request. The dynamic mass virtual hosting technique used here is based on automatically inserting this information into the pathname of the file that is used to satisfy the request. This can be most easily done by using mod_vhost_alias with Apache httpd. Alternatively, mod_rewrite can be used.

Both of these modules are disabled by default; you must enable one of them when configuring and building Apache httpd if you want to use this technique.

A couple of things need to be determined from the request in order to make the dynamic virtual host look like a normal one. The most important is the server name, which is used by the server to generate self-referential URLs etc. It is configured with the ServerName directive, and it is available to CGIs via the SERVER_NAME environment variable. The actual value used at runtime is controlled by the UseCanonicalName setting. With UseCanonicalName Off, the server name is taken from the contents of the Host: header in the request. With UseCanonicalName DNS, it is taken from a reverse DNS lookup of the virtual host's IP address. The former setting is used for name-based dynamic virtual hosting, and the latter is used for IP-based hosting. If httpd cannot work out the server name because there is no Host: header, or the DNS lookup fails, then the value configured with ServerName is used instead.

The other thing to determine is the document root (configured with DocumentRoot and available to CGI scripts via the DOCUMENT_ROOT environment variable). In a normal configuration, this is used by the core module when mapping URLs to filenames, but when the server is configured to do dynamic
virtual hosting, that job must be taken over by another module (either mod_vhost_alias or mod_rewrite), which has a different way of doing the mapping. Neither of these modules is responsible for setting the DOCUMENT_ROOT environment variable so if any CGIs or SSI documents make use of it, they will get a misleading value.
This extract from `httpd.conf` implements the virtual host arrangement outlined in the Motivation section above using `mod_vhost_alias`.

```conf
# get the server name from the Host: header
UseCanonicalName Off

# this log format can be split per-virtual-host
# using the split-logfile utility.
LogFormat "%V %h %l %u %t ""%r" %s %b" vcommon
CustomLog "logs/access_log" vcommon

# include the server name in the filenames
VirtualDocumentRoot "/www/hosts/%0/docs"
VirtualScriptAlias "/www/hosts/%0/cgi-bin"
```

This configuration can be changed into an IP-based virtual hosting solution by just turning `UseCanonicalName Off` into `UseCanonicalName DNS`. The server name that is inserted into the filename is then derived from the IP address of the virtual host. The variable `%0` references the requested servername, as indicated in the Host: header.

See the `mod_vhost_alias` documentation for more usage examples.
This is an adjustment of the above system, tailored for an ISP's web hosting server. Using %2, we can select substrings of the server name to use in the filename so that, for example, the documents for www.user.example.com are found in /home/user/www. It uses a single cgi-bin directory instead of one per virtual host.

UseCanonicalName Off
LogFormat "%V %h %l %u %t "%r" %s %b" vcommon
CustomLog logs/access_log vcommon

# include part of the server name in the filenames
VirtualDocumentRoot "/home/%2/www"

# single cgi-bin directory
ScriptAlias "/cgi-bin/" "/www/std-cgi/"

There are examples of more complicated VirtualDocumentRoot settings in the mod_vhost_alias documentation.
With more complicated setups, you can use httpd's normal `<VirtualHost>` directives to control the scope of the various virtual hosting configurations. For example, you could have one IP address for general customers' homepages, and another for commercial customers, with the following setup. This can be combined with conventional `<VirtualHost>` configuration sections, as shown below.

```html
UseCanonicalName Off

LogFormat "%V %h %l %u %t ""%r" "%s %b" vcommon

<Directory "/www/commercial">
    Options FollowSymLinks
    AllowOverride All
</Directory>

<Directory "/www/homepages">
    Options FollowSymLinks
    AllowOverride None
</Directory>

<VirtualHost 111.22.33.44>
    ServerName www.commercial.example.com

    CustomLog "logs/access_log.commercial" vcommon

    VirtualDocumentRoot "/www/commercial/%0/docs"
    VirtualScriptAlias "/www/commercial/%0/cgi-bin"
</VirtualHost>

<VirtualHost 111.22.33.45>
    ServerName www.homepages.example.com
```
Note

If the first VirtualHost block does not include a `ServerName` directive, the reverse DNS of the relevant IP will be used instead. If this is not the server name you wish to use, a bogus entry (eg. `ServerName none.example.com`) can be added to get around this behaviour.
The configuration changes suggested to turn the first example into an IP-based virtual hosting setup result in a rather inefficient setup. A new DNS lookup is required for every request. To avoid this overhead, the filesystem can be arranged to correspond to the IP addresses, instead of to the host names, thereby negating the need for a DNS lookup. Logging will also have to be adjusted to fit this system.

```apache
# get the server name from the reverse DNS of the IP address
UseCanonicalName DNS

# include the IP address in the logs so they can be split
LogFormat "%A %h %l %u %t "%r" %s %b" vcommon
CustomLog "logs/access_log" vcommon

# include the IP address in the filenames
VirtualDocumentRootIP "/www/hosts/%0/docs"
VirtualScriptAliasIP "/www/hosts/%0/cgi-bin"
```
Mass virtual hosting may also be accomplished using `mod_rewrite`, either using simple `RewriteRule` directives, or using more complicated techniques such as storing the vhost definitions externally and accessing them via `RewriteMap`. These techniques are discussed in the `rewrite documentation`.
# Apache hosts

## IP DNS

<table>
<thead>
<tr>
<th>IP</th>
<th>DNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>(CNAME)</td>
</tr>
</tbody>
</table>

## www.example.org

## DNS

- **CDN**:

## ServerName

## VirtualHost

```
# Ensure that Apache listens on port 80
Listen 80

# Listen for virtual host requests on all IP addresses
NameVirtualHost *:80

<VirtualHost *:80>
  DocumentRoot /www/example1
  ServerName www.example.com

  # Other directives here
</VirtualHost>

<VirtualHost *:80>
  DocumentRoot /www/example2
  ServerName www.example.org

  # Other directives here
</VirtualHost>
```
| IP | IP |
Listen 80

# This is the "main" server running on 172.20.30.40
ServerName server.domain.com
DocumentRoot /www/mainserver

# This is the other address
NameVirtualHost 172.20.30.50

<VirtualHost 172.20.30.50>
  DocumentRoot /www/example1
  ServerName www.example.com

  # Other directives here ...
</VirtualHost>

<VirtualHost 172.20.30.50>
  DocumentRoot /www/example2
  ServerName www.example.org

  # Other directives here ...
</VirtualHost>
VirtualHost

NameVirtualHost 192.168.1.1
NameVirtualHost 172.20.30.40

<VirtualHost 192.168.1.1 172.20.30.40>
  DocumentRoot /www/server1
  ServerName server.example.com
  ServerAlias server
</VirtualHost>
Listen 80
Listen 8080

NameVirtualHost 172.20.30.40:80
NameVirtualHost 172.20.30.40:8080

<VirtualHost 172.20.30.40:80>
  ServerName www.example.com
  DocumentRoot /www/domain-80
</VirtualHost>

<VirtualHost 172.20.30.40:8080>
  ServerName www.example.com
  DocumentRoot /www/domain-8080
</VirtualHost>

<VirtualHost 172.20.30.40:80>
  ServerName www.example.org
  DocumentRoot /www/otherdomain-80
</VirtualHost>

<VirtualHost 172.20.30.40:8080>
  ServerName www.example.org
  DocumentRoot /www/otherdomain-8080
</VirtualHost>
Listen 80

<VirtualHost 172.20.30.40>
    DocumentRoot /www/example1
    ServerName www.example.com
</VirtualHost>

<VirtualHost 172.20.30.50>
    DocumentRoot /www/example2
    ServerName www.example.org
</VirtualHost>

<VirtualHost>
    (localhost

</VirtualHost>
### IP and Listen Details

<table>
<thead>
<tr>
<th>IP</th>
<th>Listen</th>
<th>Listened Port Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>(172.20.30.40</td>
<td>172.20.30.50</td>
<td>80 8080</td>
</tr>
</tbody>
</table>

### Virtual Host Details

```xml
<VirtualHost 172.20.30.40:80>
    DocumentRoot /www/example1-80
    ServerName www.example.com
</VirtualHost>

<VirtualHost 172.20.30.40:8080>
    DocumentRoot /www/example1-8080
    ServerName www.example.com
</VirtualHost>

<VirtualHost 172.20.30.50:80>
    DocumentRoot /www/example2-80
    ServerName www.example.org
</VirtualHost>

<VirtualHost 172.20.30.50:8080>
    DocumentRoot /www/example2-8080
    ServerName www.example.org
</VirtualHost>
```
Listen 80

NameVirtualHost 172.20.30.40

<VirtualHost 172.20.30.40>
  DocumentRoot /www/example1
  ServerName www.example.com
</VirtualHost>

<VirtualHost 172.20.30.40>
  DocumentRoot /www/example2
  ServerName www.example.org
</VirtualHost>

<VirtualHost 172.20.30.40>
  DocumentRoot /www/example3
  ServerName www.example3.net
</VirtualHost>

# IP-based
<VirtualHost 172.20.30.50>
  DocumentRoot /www/example4
  ServerName www.example4.edu
</VirtualHost>

<VirtualHost 172.20.30.60>
  DocumentRoot /www/example5
  ServerName www.example5.gov
</VirtualHost>
<VirtualHost *::*>
    ProxyPreserveHost On
    ProxyPass / http://192.168.111.2/
    ProxyPassReverse / http://192.168.111.2/
    ServerName hostname.example.com
</VirtualHost>
<VirtualHost _default_:80>
  DocumentRoot /www/default80
  # ...
</VirtualHost>

<VirtualHost _default_:80>
  DocumentRoot /www/default80
  # ...
</VirtualHost>

<VirtualHost _default_:*>>
  DocumentRoot /www/default
  # ...
</VirtualHost>
<VirtualHost _default_:80>
DocumentRoot /www/default
...
</VirtualHost>
IP (172.20.30.50) VirtualHost

Listen 80
ServerName www.example.com
DocumentRoot /www/example1

NameVirtualHost 172.20.30.40

<VirtualHost 172.20.30.40 172.20.30.50>
   DocumentRoot /www/example2
   ServerName www.example.org
   # ...
</VirtualHost>

<VirtualHost 172.20.30.40>
   DocumentRoot /www/example3
   ServerName www.example.net
   ServerAlias *.example.net
   # ...
</VirtualHost>

(IP) ()
NameVirtualHost 172.20.30.40

<VirtualHost 172.20.30.40>
    # primary vhost
    DocumentRoot /www/subdomain
    RewriteEngine On
    RewriteRule ^/.* /www/subdomain/index.html
    # ...
</VirtualHost>

<VirtualHost 172.20.30.40>
    DocumentRoot /www/subdomain/sub1
    ServerName www.sub1.domain.tld
    ServerPath /sub1/
    RewriteEngine On
    RewriteRule ^(/sub1/.*) /www/subdomain$1
    # ...
</VirtualHost>

<VirtualHost 172.20.30.40>
    DocumentRoot /www/subdomain/sub2
    ServerName www.sub2.domain.tld
    ServerPath /sub2/
    RewriteEngine On
    RewriteRule ^(/sub2/.*) /www/subdomain$1
    # ...
</VirtualHost>
An In-Depth Discussion of Virtual Host Matching

This document attempts to explain exactly what Apache HTTP Server does when deciding what virtual host to serve a request from.

Most users should read about Name-based vs. IP-based Virtual Hosts to decide which type they want to use, then read more about name-based or IP-based virtualhosts, and then see some examples.

If you want to understand all the details, then you can come back to this page.

See also

IP-based Virtual Host Support
Name-based Virtual Hosts Support
Virtual Host examples for common setups
Dynamically configured mass virtual hosting
There is a main server which consists of all the definitions appearing outside of `<VirtualHost>` sections.

There are virtual servers, called vhosts, which are defined by `<VirtualHost>` sections.

Each `VirtualHost` directive includes one or more addresses and optional ports.

Hostnames can be used in place of IP addresses in a virtual host definition, but they are resolved at startup and if any name resolutions fail, those virtual host definitions are ignored. This is, therefore, not recommended.

The address can be specified as `*`, which will match a request if no other vhost has the explicit address on which the request was received.

The address appearing in the `VirtualHost` directive can have an optional port. If the port is unspecified, it is treated as a wildcard port, which can also be indicated explicitly using `*`. The wildcard port matches any port.

(Port numbers specified in the `VirtualHost` directive do not influence what port numbers Apache will listen on, they only control which `VirtualHost` will be selected to handle a request. Use the `Listen` directive to control the addresses and ports on which the server listens.)

Collectively the entire set of addresses (including multiple results from DNS lookups) are called the vhost's address set.

Apache automatically discriminates on the basis of the HTTP Host header supplied by the client whenever the most specific
match for an IP address and port combination is listed in multiple virtual hosts.

The **ServerName** directive may appear anywhere within the definition of a server. However, each appearance overrides the previous appearance (within that server). If no **ServerName** is specified, the server attempts to deduce it from the server's IP address.

The first name-based vhost in the configuration file for a given IP:port pair is significant because it is used for all requests received on that address and port for which no other vhost for that IP:port pair has a matching **ServerName** or **ServerAlias**. It is also used for all SSL connections if the server does not support **Server Name Indication**.

The complete list of names in the **VirtualHost** directive are treated just like a (non wildcard) **ServerAlias** (but are not overridden by any **ServerAlias** statement).

For every vhost various default values are set. In particular:

1. If a vhost has no **ServerAdmin**, **Timeout**, **KeepAliveTimeout**, **KeepAlive**, **MaxKeepAliveRequests**, **ReceiveBufferSize**, or **SendBufferSize** directive then the respective value is inherited from the main server. (That is, inherited from whatever the final setting of that value is in the main server.)

2. The "lookup defaults" that define the default directory permissions for a vhost are merged with those of the main server. This includes any per-directory configuration information for any module.

3. The per-server configs for each module from the main server are merged into the vhost server.
Essentially, the main server is treated as "defaults" or a "base" on which to build each vhost. But the positioning of these main server definitions in the config file is largely irrelevant -- the entire config of the main server has been parsed when this final merging occurs. So even if a main server definition appears after a vhost definition it might affect the vhost definition.

If the main server has no ServerName at this point, then the hostname of the machine that httpd is running on is used instead. We will call the main server address set those IP addresses returned by a DNS lookup on the ServerName of the main server.

For any undefined ServerName fields, a name-based vhost defaults to the address given first in the VirtualHost statement defining the vhost.

Any vhost that includes the magic _default_ wildcard is given the same ServerName as the main server.
The server determines which vhost to use for a request as follows:

**IP address lookup**

When the connection is first received on some address and port, the server looks for all the VirtualHost definitions that have the same IP address and port.

If there are no exact matches for the address and port, then wildcard (*) matches are considered.

If no matches are found, the request is served by the main server.

If there are VirtualHost definitions for the IP address, the next step is to decide if we have to deal with an IP-based or a name-based vhost.

**IP-based vhost**

If there is exactly one VirtualHost directive listing the IP address and port combination that was determined to be the best match, no further actions are performed and the request is served from the matching vhost.

**Name-based vhost**

If there are multiple VirtualHost directives listing the IP address and port combination that was determined to be the best match, the "list" in the remaining steps refers to the list of vhosts that matched, in the order they were in the configuration file.

If the connection is using SSL, the server supports Server Name Indication, and the SSL client handshake includes the TLS extension with the requested hostname, then that hostname is used below just like the Host: header would be used on a non-
SSL connection. Otherwise, the first name-based vhost whose address matched is used for SSL connections. This is significant because the vhost determines which certificate the server will use for the connection.

If the request contains a Host: header field, the list is searched for the first vhost with a matching ServerName or ServerAlias, and the request is served from that vhost. A Host: header field can contain a port number, but Apache always ignores it and matches against the real port to which the client sent the request.

The first vhost in the config file with the specified IP address has the highest priority and catches any request to an unknown server name, or a request without a Host: header field (such as a HTTP/1.0 request).

**Persistent connections**

The *IP lookup* described above is only done *once* for a particular TCP/IP session while the *name lookup* is done on *every* request during a KeepAlive/persistent connection. In other words, a client may request pages from different name-based vhosts during a single persistent connection.

**Absolute URI**

If the URI from the request is an absolute URI, and its hostname and port match the main server or one of the configured virtual hosts *and* match the address and port to which the client sent the request, then the scheme/hostname/port prefix is stripped off and the remaining relative URI is served by the corresponding main server or virtual host. If it does not match, then the URI remains untouched and the request is taken to be a proxy request.

**Observations**
- Name-based virtual hosting is a process applied after the server has selected the best matching IP-based virtual host.
- If you don't care what IP address the client has connected to, use a "*" as the address of every virtual host, and name-based virtual hosting is applied across all configured virtual hosts.
- ServerName and ServerAlias checks are never performed for an IP-based vhost.
- Only the ordering of name-based vhosts for a specific address set is significant. The one name-based vhost that comes first in the configuration file has the highest priority for its corresponding address set.
- Any port in the Host: header field is never used during the matching process. Apache always uses the real port to which the client sent the request.
- If two vhosts have an address in common, those common addresses act as name-based virtual hosts implicitly. This is new behavior as of 2.3.11.
- The main server is only used to serve a request if the IP address and port number to which the client connected does not match any vhost (including a * vhost). In other words, the main server only catches a request for an unspecified address/port combination (unless there is a _default_ vhost which matches that port).
- You should never specify DNS names in VirtualHost directives because it will force your server to rely on DNS to boot. Furthermore it poses a security threat if you do not control the DNS for all the domains listed. There's more information available on this and the next two topics.
- ServerName should always be set for each vhost. Otherwise a DNS lookup is required for each vhost.
In addition to the tips on the DNS Issues page, here are some further tips:

- Place all main server definitions before any VirtualHost definitions. (This is to aid the readability of the configuration -- the post-config merging process makes it non-obvious that definitions mixed in around virtual hosts might affect all virtual hosts.)
Apache

1. setrlimit()
2. setrlimit(RLIMIT_NOFILE) (Solaris 2.3)
3.
4. stdio 256

: 

- `<VirtualHost>`
- 1 2 Apache

```sh
#!/bin/sh
ulimit -S -n 100
exec httpd
```
LogFormat  %v : 

LogFormat "%v %h %l %u %t "%r" %>s %b" vhost
CustomLog logs/multiple_vhost_log vhost

common log format ( ServerName ) ( Custom Log Formats )

(1)  split-logfile  Apache support

: 

split-logfile < /logs/multiple_vhost_log

.log
<table>
<thead>
<tr>
<th>DNS</th>
<th>Apache</th>
<th>DNS A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ServerName</td>
<td>ServerAdmin</td>
<td>DocumentRoot</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
<td>---------------</td>
</tr>
<tr>
<td><a href="http://www.abc.dom">www.abc.dom</a></td>
<td><a href="mailto:webgirl@abc.dom">webgirl@abc.dom</a></td>
<td>/www/abc</td>
</tr>
<tr>
<td><a href="http://www.abc.dom">www.abc.dom</a></td>
<td><a href="mailto:webgirl@abc.dom">webgirl@abc.dom</a></td>
<td>/www/abc</td>
</tr>
</tbody>
</table>

Apache DNS

<table>
<thead>
<tr>
<th>IP</th>
<th>URL</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
<VirtualHost www.abc.dom>
    ServerAdmin webgirl@abc.dom
    DocumentRoot /www/abc
</VirtualHost>

<VirtualHost www.def.dom>
    ServerAdmin webguy@def.dom
    DocumentRoot /www/def
</VirtualHost>

192.0.2.1  www.abc.dom  192.0.2.2  www.def.dom
    def.dom DNS
www.def.dom  192.0.2.1 DNS DNS
www.def.dom

192.0.2.1 (http://www.abc.dom/whatever URL
def.dom

//
- `VirtualHost` IP
- `Listen` IP
- `ServerName`
- `<VirtualHost _default_:*>`
SSL/TLS:

-- A. Tanenbaum, "Introduction to Computer Networks"

WebHTTPApache SSL

mod_ssl

The Open Group Research Institute Frederick J. Hirsch
Introducing SSL and Certificates using SSLeay Web
Engelschall (mod_ssl) (: Apache )
SSL (:) ([ AC96] )
(Certificate Authority  CA)  
(certificate) (authentication)

1:

2:

<table>
<thead>
<tr>
<th>Common Name ()</th>
<th>CN SSL URL</th>
<th>CN=www.example.com</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization or Company ()</td>
<td>O</td>
<td>O=Example Japan K.K.</td>
</tr>
<tr>
<td>Organizational Unit ()</td>
<td>OU</td>
<td>OU=Customer Service</td>
</tr>
<tr>
<td>City/Locality ()</td>
<td>L</td>
<td>L=Sapporo</td>
</tr>
<tr>
<td>State/Province ()</td>
<td>ST</td>
<td>ST=Hokkaido</td>
</tr>
<tr>
<td>Country()</td>
<td>C</td>
<td>ISO C=JP</td>
</tr>
</tbody>
</table>
ASN.1 Encoding Rules (DER) [X208] [PKCS]
Basic Encoding Rules [MIME] [ASCII]

Base64 [MIME] ASCII

PEM (example.crt)

-----BEGIN CERTIFICATE-----
MIIC7jCCAxAgIBAgIBATANBgkqhkiG9w0BAQQFADBqTELMaKGA1UEBhMCWFkx
FTATBgNVBAgTDFNtYyt1IER1c2VyYdDETBMEGA1UEBxMKU25ha2U6G93bjEXMBUG
A1UEChM0U25ha2U6G93bjEXMBUG
asn.1

-----END CERTIFICATE-----
### Secure Sockets Layer (TCP/IP) (HTTP) SSL

SSL

#### 4: SSL

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
<th>Protocol</th>
<th>Supported Applications</th>
</tr>
</thead>
</table>
| SSL 2.0 | Vendor Standard (Netscape Corp.) | SSL | - NS Navigator 1.x/2.x  
- MS IE 3.x  
- Lynx/2.8+OpenSSL |
| SSL 3.0 | Expired Internet Draft (Netscape Corp.) | SSL | - NS Navigator 2.x/3.x/4.x  
- MS IE 3.x/4.x  
- Lynx/2.8+OpenSSL |
| TLS 1.0 | Proposed Internet Standard (IETF) | MAC HMAC block padding 3.0 | - Lynx/2.8+OpenSSL |

4SSL  SSL 3.0  
SSL 3.0  Internet Engineering Task Force (IETF)  
Transport Layer Security [TLS]
1: SSL

- Establish protocol version, session id, cipher suite, compression method
- Exchange random values
- Optionally send server certificate and request client certificate
- Send client certificate response if requested
- Change CipherSpec and Finish Handshake

1. 
2. 
3. 
4.

- Message Authentication Code (MAC)

Diffie-Hellman
SSL

- 40-bit RC4
- 128-bit RC4

- CBC
  - 40 bit RC2
  - 40 bit DES
  - 56 bit DES
  - 168 bit Triple-DES
  - Idea (128 bit)
  - Fortezza (96 bit)

CBC (Cipher Block Chaining) Encryption Standard) [AC96, ch12] DES40 3DES_EDE Idea

- MD5 (128-bit)
- Secure Hash Algorithm (SHA-1) (160-bit)

Message Authentication Code (MAC)
• SSL
• SSL
• SSL

2: SSL

SSL

3 SSL

3 : SSL
<table>
<thead>
<tr>
<th>HTTP</th>
<th>SSL HTTP</th>
<th>URL</th>
<th>http</th>
<th>https (443)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
[AC96]

[X208]

[X509]

[PKCS]

[MIME]

[SSL2]

[SSL3]
[TLS1]
This page covers backwards compatibility between mod_ssl and other SSL solutions. mod_ssl is not the only SSL solution for Apache; four additional products are (or were) also available: Ben Laurie's freely available Apache-SSL (from where mod_ssl were originally derived in 1998), Red Hat's commercial Secure Web Server (which was based on mod_ssl), Covalent's commercial Raven SSL Module (also based on mod_ssl) and finally C2Net's (now Red Hat's) commercial product Stronghold (based on a different evolution branch, named Sioux up to Stronghold 2.x, and based on mod_ssl since Stronghold 3.x).

mod_ssl mostly provides a superset of the functionality of all the other solutions, so it's simple to migrate from one of the older modules to mod_ssl. The configuration directives and environment variable names used by the older SSL solutions vary from those used in mod_ssl; mapping tables are included here to give the equivalents used by mod_ssl.
The mapping between configuration directives used by Apache-SSL 1.x and mod_ssl 2.0.x is given in Table 1. The mapping from Sioux 1.x and Stronghold 2.x is only partial because of special functionality in these interfaces which mod_ssl doesn't provide.

### Table 1: Configuration Directive Mapping

<table>
<thead>
<tr>
<th>Old Directive</th>
<th>mod_ssl Directive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Apache-SSL 1.x &amp; mod_ssl 2.0.x compatibility:</strong></td>
<td></td>
</tr>
<tr>
<td>SSLEnable</td>
<td>SSLEngine on</td>
</tr>
<tr>
<td>SSLDisable</td>
<td>SSLEngine off</td>
</tr>
<tr>
<td>SSLLogFile file</td>
<td></td>
</tr>
<tr>
<td>SSLRequiredCiphers spec</td>
<td>SSLCipherSuite spec</td>
</tr>
<tr>
<td>SSLRequireCipher c1 ...</td>
<td>SSLRequire %{SSL_CIPHER} in &quot;c1&quot;, ...</td>
</tr>
<tr>
<td>SSLBanCipher c1 ...</td>
<td>SSLRequire not (%{SSL_CIPHER} in &quot;c1&quot;, ...}</td>
</tr>
<tr>
<td>SSLFakeBasicAuth</td>
<td>SSLOptions +FakeBasicAuth</td>
</tr>
<tr>
<td>SSLCacheServerPath dir</td>
<td></td>
</tr>
<tr>
<td>SSLCacheServerPort integer</td>
<td></td>
</tr>
<tr>
<td><strong>Apache-SSL 1.x compatibility:</strong></td>
<td></td>
</tr>
<tr>
<td>SSLExportClientCertificates</td>
<td>SSLOptions +ExportCert</td>
</tr>
<tr>
<td>SSLCacheServerRunDir dir</td>
<td></td>
</tr>
<tr>
<td><strong>Sioux 1.x compatibility:</strong></td>
<td></td>
</tr>
<tr>
<td>SSL_CertFile file</td>
<td>SSLCertificateFile file</td>
</tr>
<tr>
<td>SSL_KeyFile file</td>
<td>SSLCertificateKeyFile</td>
</tr>
<tr>
<td>SSL_CipherSuite arg</td>
<td>SSLCipherSuite arg</td>
</tr>
<tr>
<td>SSL_X509VerifyDir arg</td>
<td>SSLCACertificatePath arg</td>
</tr>
<tr>
<td>----------------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>SSL_Log file</td>
<td>-</td>
</tr>
<tr>
<td>SSL_Connect flag</td>
<td>SSLEngine flag</td>
</tr>
<tr>
<td>SSL_ClientAuth arg</td>
<td>SSLVerifyClient arg</td>
</tr>
<tr>
<td>SSL_X509VerifyDepth arg</td>
<td>SSLVerifyDepth arg</td>
</tr>
<tr>
<td>SSL_FetchKeyPhraseFrom arg</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SessionDir dir</td>
<td>-</td>
</tr>
<tr>
<td>SSL_Require expr</td>
<td>-</td>
</tr>
<tr>
<td>SSL_CertFileType arg</td>
<td>-</td>
</tr>
<tr>
<td>SSL_KeyFileType arg</td>
<td>-</td>
</tr>
<tr>
<td>SSL_X509VerifyPolicy arg</td>
<td>-</td>
</tr>
<tr>
<td>SSL_LogX509Attributes arg</td>
<td>-</td>
</tr>
</tbody>
</table>

**Stronghold 2.x compatibility:**

<table>
<thead>
<tr>
<th>StrongholdAccelerator engine</th>
<th>SSLCryptoDevice engine</th>
</tr>
</thead>
<tbody>
<tr>
<td>StrongholdKey dir</td>
<td>-</td>
</tr>
<tr>
<td>StrongholdLicenseFile dir</td>
<td>-</td>
</tr>
<tr>
<td>SSLFlag flag</td>
<td>SSLEngine flag</td>
</tr>
<tr>
<td>SSLSessionLockFile file</td>
<td>SSLMutex file</td>
</tr>
<tr>
<td>SSLCipherList spec</td>
<td>SSLCipherSuite spec</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>RequireSSL</td>
<td>SSLRequireSSL</td>
</tr>
<tr>
<td>SSLErrorFile file</td>
<td>-</td>
</tr>
<tr>
<td>SSLRoot dir</td>
<td>-</td>
</tr>
<tr>
<td>SSL_CertificateLogDir dir</td>
<td>-</td>
</tr>
<tr>
<td>AuthCertDir dir</td>
<td>-</td>
</tr>
<tr>
<td>SSL_Group name</td>
<td>-</td>
</tr>
<tr>
<td>SSLProxyMachineCertPath dir</td>
<td>SSLProxyMachineCertPath dir</td>
</tr>
<tr>
<td>SSLProxyMachineCertFile file</td>
<td>SSLProxyMachineCertFile file</td>
</tr>
<tr>
<td>SSLProxyCipherList spec</td>
<td>SSLProxyCipherSpec spec</td>
</tr>
</tbody>
</table>
The mapping between environment variable names used by the older SSL solutions and the names used by mod_ssl is given in Table 2.

**Table 2: Environment Variable Derivation**

<table>
<thead>
<tr>
<th>Old Variable</th>
<th>mod_ssl Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_PROTOCOL_VERSION</td>
<td>SSL_PROTOCOL</td>
</tr>
<tr>
<td>SSLEAY_VERSION</td>
<td>SSL_VERSION_LIBRARY</td>
</tr>
<tr>
<td>HTTPS_SECRETKEYSIZE</td>
<td>SSL_CIPHER_USEKEYSIZE</td>
</tr>
<tr>
<td>HTTPS_KEYSIZE</td>
<td>SSL_CIPHER_ALGKEYS</td>
</tr>
<tr>
<td>HTTPS_CIPHER</td>
<td>SSL_CIPHER</td>
</tr>
<tr>
<td>HTTPS_EXPORT</td>
<td>SSL_CIPHER_EXPORT</td>
</tr>
<tr>
<td>SSL_SERVER_KEY_SIZE</td>
<td>SSL_CIPHER_ALGKEYS</td>
</tr>
<tr>
<td>SSL_SERVER_CERTIFICATE</td>
<td>SSL_SERVER_CERT</td>
</tr>
<tr>
<td>SSL_SERVER_CERT_START</td>
<td>SSL_SERVER_V_START</td>
</tr>
<tr>
<td>SSL_SERVER_CERT_END</td>
<td>SSL_SERVER_V_END</td>
</tr>
<tr>
<td>SSL_SERVER_CERT_SERIAL</td>
<td>SSL_SERVER_M_SERIAL</td>
</tr>
<tr>
<td>SSL_SERVER_SIGNATURE_ALGORITHM</td>
<td>SSL_SERVER_A_SIG</td>
</tr>
<tr>
<td>SSL_SERVER_DN</td>
<td>SSL_SERVER_S_DN</td>
</tr>
<tr>
<td>SSL_SERVER_CN</td>
<td>SSL_SERVER_S_DN_CN</td>
</tr>
<tr>
<td>SSL_SERVER_EMAIL</td>
<td>SSL_SERVER_S_DN_Email</td>
</tr>
<tr>
<td>SSL_SERVER_O</td>
<td>SSL_SERVER_S_DN_O</td>
</tr>
<tr>
<td>SSL_SERVER_OU</td>
<td>SSL_SERVER_S_DN_OU</td>
</tr>
<tr>
<td>SSL_SERVER_C</td>
<td>SSL_SERVER_S_DN_C</td>
</tr>
<tr>
<td>SSL_SERVER_SP</td>
<td>SSL_SERVER_S_DN_SP</td>
</tr>
<tr>
<td>SSL_SERVER_L</td>
<td>SSL_SERVER_S_DN_L</td>
</tr>
<tr>
<td>SSL_SERVER_IDN</td>
<td>SSL_SERVER_I_DN</td>
</tr>
<tr>
<td>SSL_SERVER_ICN</td>
<td>SSL_SERVER_I_DN_CN</td>
</tr>
<tr>
<td>SSL_SERVER_IEMAIL</td>
<td>SSL_SERVER_I_DN_Email</td>
</tr>
<tr>
<td>SSL_SERVER_IO</td>
<td>SSL_SERVER_I_DN_O</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------</td>
</tr>
<tr>
<td>SSL_SERVER_IOU</td>
<td>SSL_SERVER_I_DN_OU</td>
</tr>
<tr>
<td>SSL_SERVER_IC</td>
<td>SSL_SERVER_I_DN_C</td>
</tr>
<tr>
<td>SSL_SERVER_ISP</td>
<td>SSL_SERVER_I_DN_SP</td>
</tr>
<tr>
<td>SSL_SERVER_IL</td>
<td>SSL_SERVER_I_DN_L</td>
</tr>
<tr>
<td>SSL_CLIENT_CERTIFICATE</td>
<td>SSL_CLIENT_CERT</td>
</tr>
<tr>
<td>SSL_CLIENT_CERT_START</td>
<td>SSL_CLIENT_V_START</td>
</tr>
<tr>
<td>SSL_CLIENT_CERT_END</td>
<td>SSL_CLIENT_V_END</td>
</tr>
<tr>
<td>SSL_CLIENT_CERT_SERIAL</td>
<td>SSL_CLIENT_M_SERIAL</td>
</tr>
<tr>
<td>SSL_CLIENT_SIGNATURE_ALGORITHM</td>
<td>SSL_CLIENT_A_SIG</td>
</tr>
<tr>
<td>SSL_CLIENT_DN</td>
<td>SSL_CLIENT_S_DN</td>
</tr>
<tr>
<td>SSL_CLIENT_CN</td>
<td>SSL_CLIENT_S_DN_CN</td>
</tr>
<tr>
<td>SSL_CLIENT_EMAIL</td>
<td>SSL_CLIENT_S_DN_Email</td>
</tr>
<tr>
<td>SSL_CLIENT_O</td>
<td>SSL_CLIENT_S_DN_O</td>
</tr>
<tr>
<td>SSL_CLIENT_OU</td>
<td>SSL_CLIENT_S_DN_OU</td>
</tr>
<tr>
<td>SSL_CLIENT_C</td>
<td>SSL_CLIENT_S_DN_C</td>
</tr>
<tr>
<td>SSL_CLIENT_SP</td>
<td>SSL_CLIENT_S_DN_SP</td>
</tr>
<tr>
<td>SSL_CLIENT_L</td>
<td>SSL_CLIENT_S_DN_L</td>
</tr>
<tr>
<td>SSL_CLIENT_IDN</td>
<td>SSL_CLIENT_I_DN</td>
</tr>
<tr>
<td>SSL_CLIENT_ICN</td>
<td>SSL_CLIENT_I_DN_CN</td>
</tr>
<tr>
<td>SSL_CLIENT_IEMAIL</td>
<td>SSL_CLIENT_I_DN_Email</td>
</tr>
<tr>
<td>SSL_CLIENT_IO</td>
<td>SSL_CLIENT_I_DN_O</td>
</tr>
<tr>
<td>SSL_CLIENT_IOU</td>
<td>SSL_CLIENT_I_DN_OU</td>
</tr>
<tr>
<td>SSL_CLIENT_IC</td>
<td>SSL_CLIENT_I_DN_C</td>
</tr>
<tr>
<td>SSL_CLIENT_ISP</td>
<td>SSL_CLIENT_I_DN_SP</td>
</tr>
<tr>
<td>SSL_CLIENT_IL</td>
<td>SSL_CLIENT_I_DN_L</td>
</tr>
<tr>
<td>SSL_EXPORT</td>
<td>SSL_CIPHER_EXPORT</td>
</tr>
<tr>
<td>SSL_KEYSIZE</td>
<td>SSL_CIPHER_ALGKEYSIZE</td>
</tr>
<tr>
<td>SSL_SECKEYSIZE</td>
<td>SSL_CIPHER_USEKEYSIZE</td>
</tr>
</tbody>
</table>
| SSL_SSLEAY_VERSION     | SSL_VERSION_LIBRAR'
<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL_STRONG_CRYPTO</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SERVER_KEY_EXP</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SERVER_KEY_ALGORITHM</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SERVER_KEY_SIZE</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SERVER_SESSIONDIR</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SERVER_CERTIFICATELOGDIR</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SERVER_CERTFILE</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SERVER_KEYFILE</td>
<td>-</td>
</tr>
<tr>
<td>SSL_SERVER_KEYFILETYPE</td>
<td>-</td>
</tr>
<tr>
<td>SSL_CLIENT_KEY_EXP</td>
<td>-</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----</td>
</tr>
<tr>
<td>SSL_CLIENT_KEY_ALGORITHM</td>
<td>-</td>
</tr>
<tr>
<td>SSL_CLIENT_KEY_SIZE</td>
<td>-</td>
</tr>
</tbody>
</table>
When `mod_ssl` is enabled, additional functions exist for the Custom Log Format of `mod_log_config` as documented in the Reference Chapter. Beside the ``%{varname}x`` eXtension format function which can be used to expand any variables provided by any module, an additional Cryptography ``%{name}c`` cryptography format function exists for backward compatibility. The currently implemented function calls are listed in Table 3.

**Table 3: Custom Log Cryptography Function**

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%{version}c</td>
<td>SSL protocol version</td>
</tr>
<tr>
<td>%{cipher}c</td>
<td>SSL cipher</td>
</tr>
<tr>
<td>%{subjectdn}c</td>
<td>Client Certificate Subject Distinguished Name</td>
</tr>
<tr>
<td>%{issuerdn}c</td>
<td>Client Certificate Issuer Distinguished Name</td>
</tr>
<tr>
<td>%{errcode}c</td>
<td>Certificate Verification Error (numerical)</td>
</tr>
<tr>
<td>%{errstr}c</td>
<td>Certificate Verification Error (string)</td>
</tr>
</tbody>
</table>
SSL/TLS Strong Encryption: How-To

This document is intended to get you started, and get a few things working. You are strongly encouraged to read the rest of the SSL documentation, and arrive at a deeper understanding of the material, before progressing to the advanced techniques.
Your SSL configuration will need to contain, at minimum, the following directives.

```
LoadModule ssl_module modules/mod_ssl.so

Listen 443
<VirtualHost *:443>
  ServerName www.example.com
  SSLEngine on
  SSLCertificateFile "/path/to/www.example.com.cert"
  SSLCertificateKeyFile "/path/to/www.example.com.key"
</VirtualHost>
```
How can I create an SSL server which accepts strong encryption only?

The following enables only the strongest ciphers:

```
SSLCipherSuite HIGH:!aNULL:!MD5
```

While with the following configuration you specify a preference for specific speed-optimized ciphers (which will be selected by mod_ssl, provided that they are supported by the client):

```
SSLCipherSuite RC4-SHA:AES128-SHA:HIGH:!aNULL
SSLHonorCipherOrder on
```

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

Obviously, a server-wide `SSLCipherSuite` which restricts ciphers to the strong variants, isn't the answer here. However, `mod_ssl` can be reconfigured within Location blocks, to give a per-directory solution, and can automatically force a renegotiation of the SSL parameters to meet the new configuration. This can be done as follows:

```
# be liberal in general
```

<Location "/strong/area">
# but https://hostname/strong/area/ and below
# requires strong ciphers
SSLCipherSuite HIGH:!aNULL:!MD5
</Location>
The Online Certificate Status Protocol (OCSP) is a mechanism for determining whether or not a server certificate has been revoked, and OCSP Stapling is a special form of this in which the server, such as httpd and mod_ssl, maintains current OCSP responses for its certificates and sends them to clients which communicate with the server. Most certificates contain the address of an OCSP responder maintained by the issuing Certificate Authority, and mod_ssl can communicate with that responder to obtain a signed response that can be sent to clients communicating with the server.

Because the client can obtain the certificate revocation status from the server, without requiring an extra connection from the client to the Certificate Authority, OCSP Stapling is the preferred way for the revocation status to be obtained. Other benefits of eliminating the communication between clients and the Certificate Authority are that the client browsing history is not exposed to the Certificate Authority and obtaining status is more reliable by not depending on potentially heavily loaded Certificate Authority servers.

Because the response obtained by the server can be reused for all clients using the same certificate during the time that the response is valid, the overhead for the server is minimal.

Once general SSL support has been configured properly, enabling OCSP Stapling generally requires only very minor modifications to the httpd configuration — the addition of these two directives:

```
SSLUseStapling On
SSLStaplingCache "shmcb:logs/ssl_stapling(32768)
```

These directives are placed at global scope (i.e., not within a virtual host definition) wherever other global SSL configuration
directives are placed, such as in conf/extra/httpd-ssl.conf for normal open source builds of httpd, /etc/apache2/mods-enabled/ssl.conf for the Ubuntu or Debian-bundled httpd, etc.

The path on the SSLStaplingCache directive (e.g., logs/) should match the one on the SSLSessionCache directive. This path is relative to ServerRoot.

This particular SSLStaplingCache directive requires mod_socache_shmcb (from the shmcb prefix on the directive's argument). This module is usually enabled already for SSLSessionCache or on behalf of some module other than mod_ssl. If you enabled an SSL session cache using a mechanism other than mod_socache_shmcb, use that alternative mechanism for SSLStaplingCache as well. For example:

```
SSLSessionCache "dbm:logs/ssl_scache"
SSLStaplingCache "dbm:logs/ssl_stapling"
```

You can use the openssl command-line program to verify that an OCSP response is sent by your server:

```
$ openssl s_client -connect www.example.com:443 -v
... OCSP response:
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
... Cert Status: Good
...```

The following sections highlight the most common situations which require further modification to the configuration. Refer also to the
mod_ssl reference manual.

If more than a few SSL certificates are used for the server

OCSP responses are stored in the SSL stapling cache. While the responses are typically a few hundred to a few thousand bytes in size, mod_ssl supports OCSP responses up to around 10K bytes in size. With more than a few certificates, the stapling cache size (32768 bytes in the example above) may need to be increased. Error message AH01929 will be logged in case of an error storing a response.

If the certificate does not point to an OCSP responder, or if a different address must be used

Refer to the SSLStaplingForceURL directive.

You can confirm that a server certificate points to an OCSP responder using the openssl command-line program, as follows:

$ openssl x509 -in ./www.example.com.crt -text | grep 'OCSP.*http'
OCSP - URI:http://ocsp.example.com

If the OCSP URI is provided and the web server can communicate to it directly without using a proxy, no configuration is required. Note that firewall rules that control outbound connections from the web server may need to be adjusted.

If no OCSP URI is provided, contact your Certificate Authority to determine if one is available; if so, configure it with SSLStaplingForceURL in the virtual host that uses the certificate.

If multiple SSL-enabled virtual hosts are configured and OCSP Stapling should be disabled for some
Add SSLUseStapling Off to the virtual hosts for which OCSP Stapling should be disabled.

If the OCSP responder is slow or unreliable
Several directives are available to handle timeouts and errors. Refer to the documentation for the SSLStaplingFakeTryLater, SSLStaplingResponderTimeout, and SSLStaplingReturnResponderErrors directives.

If mod_ssl logs error AH02217
AH02217: ssl_stapling_init_cert: Can't retrieve issuer certificate!

In order to support OCSP Stapling when a particular server certificate is used, the certificate chain for that certificate must be configured. If it was not configured as part of enabling SSL, the AH02217 error will be issued when stapling is enabled, and an OCSP response will not be provided for clients using the certificate.

Refer to the SSLCertificateChainFile and SSLCertificateFile for instructions for configuring the certificate chain.
- How can I force clients to authenticate using certificates?
- How can I force clients to authenticate using certificates for a particular URL, but still allow arbitrary clients to access the rest of the server?
- How can I allow only clients who have certificates to access a particular URL, but allow all clients to access the rest of the server?
- How can I require HTTPS with strong ciphers, and either basic authentication or client certificates, for access to part of the Intranet website, for clients coming from the Internet?

### How can I force clients to authenticate using certificates?

When you know all of your users (eg, as is often the case on a corporate Intranet), you can require plain certificate authentication. All you need to do is to create client certificates signed by your own CA certificate (ca.crt) and then verify the clients against this certificate.

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

### How can I force clients to authenticate using certificates for a particular URL, but still allow arbitrary clients to access the rest of the server?

To force clients to authenticate using certificates for a particular URL, you can use the per-directory reconfiguration features of mod_ssl:
How can I allow only clients who have certificates to access a particular URL, but allow all clients to access the rest of the server?

The key to doing this is checking that part of the client certificate matches what you expect. Usually this means checking all or part of the Distinguished Name (DN), to see if it contains some known string. There are two ways to do this, using either \texttt{mod\_auth\_basic} or \texttt{SSLRequire}.

The \texttt{mod\_auth\_basic} method is generally required when the certificates are completely arbitrary, or when their DNs have no common fields (usually the organisation, etc.). In this case, you should establish a password database containing \textit{all} clients allowed, as follows:

```plaintext
SSLVerifyClient none
SSLCACertificateFile "conf/ssl.crt/ca.crt"

<Directory "/usr/local/apache2/htdocs/secure">
  SSLVerifyClient require
  SSLVerifyDepth 1
</Directory>
```
The password used in this example is the DES encrypted string "password". See the [SSLOptions](https://httpd.apache.org/docs) docs for more information.

### httpd.passwd

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

When your clients are all part of a common hierarchy, which is encoded into the DN, you can match them more easily using [SSLRequire](https://httpd.apache.org/docs), as follows:

```
SSLVerifyClient none
SSLCA CertificateFile "conf/ssl.crt/ca.crt"
SSLCA CertificatePath "conf/ssl.crt"

<Directory "/usr/local/apache2/htdocs/secure/area"
  SSLVerifyClient require
  SSLVerifyDepth 5
  SSLOptions +FakeBasicAuth
  SSLRequireSSL
  SSLRequire %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd." and %{SSL_CLIENT_S_DN_OU} in {"Staff","CA","Dev"}
</Directory>
```

How can I require HTTPS with strong ciphers, and either basic authentication or client certificates, for access to part of the Intranet website, for clients
coming from the Internet? I still want to allow plain HTTP access for clients on the Intranet.

These examples presume that clients on the Intranet have IPs in the range 192.168.1.0/24, and that the part of the Intranet website you want to allow internet access to is /usr/local/apache2/htdocs/subarea. This configuration should remain outside of your HTTPS virtual host, so that it applies to both HTTPS and HTTP.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSLCACertificateFile</td>
<td>&quot;conf/ssl.crt/company-ca.crt&quot;</td>
</tr>
<tr>
<td>&lt;Directory &quot;/usr/local/apache2/htdocs&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td># Outside the subarea only Intranet access is granted</td>
<td></td>
</tr>
<tr>
<td>Require</td>
<td>ip 192.168.1.0/24</td>
</tr>
<tr>
<td>&lt;/Directory&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;Directory &quot;/usr/local/apache2/htdocs/subarea&quot;&gt;</td>
<td></td>
</tr>
<tr>
<td># Inside the subarea any Intranet access is allowed</td>
<td></td>
</tr>
<tr>
<td># but from the Internet only HTTPS + Strong-Cipher or the alternative HTTPS + Client-Certificate</td>
<td></td>
</tr>
<tr>
<td># If HTTPS is used, make sure a strong cipher is used</td>
<td></td>
</tr>
<tr>
<td># Additionally allow client certs as an alternative</td>
<td></td>
</tr>
<tr>
<td>SSLVerifyClient</td>
<td>optional</td>
</tr>
<tr>
<td>SSLVerifyDepth</td>
<td>1</td>
</tr>
<tr>
<td>SSLOptions</td>
<td>+FakeBasicAuth +StrictRequire</td>
</tr>
<tr>
<td>SSLRequire</td>
<td>%{SSL_CIPHER_USEKEYSIZE} &gt;= 128</td>
</tr>
<tr>
<td># Force clients from the Internet to use HTTPS</td>
<td></td>
</tr>
<tr>
<td>RewriteEngine</td>
<td>on</td>
</tr>
<tr>
<td>RewriteCond</td>
<td>&quot;%{REMOTE_ADDR}&quot; &quot;!^192.(.</td>
</tr>
<tr>
<td>RewriteCond</td>
<td>&quot;%{HTTPS}&quot; &quot;!=on&quot;</td>
</tr>
<tr>
<td>RewriteRule</td>
<td>&quot;.&quot; &quot;-&quot; [F]</td>
</tr>
<tr>
<td># Allow Network Access and/or Basic Auth</td>
<td></td>
</tr>
<tr>
<td>Satisfy</td>
<td>any</td>
</tr>
</tbody>
</table>
# Network Access Control
Require ip 192.168.1.0/24

# HTTP Basic Authentication
AuthType basic
AuthName "Protected Intranet Area"
AuthBasicProvider file
AuthUserFile "conf/protected.passwd"
Require valid-user
</Directory>
mod_ssl can log extremely verbose debugging information to the error log, when its LogLevel is set to the higher trace levels. On the other hand, on a very busy server, level info may already be too much. Remember that you can configure the LogLevel per module to suite your needs.
SSL/TLS Strong Encryption: FAQ

*The wise man doesn't give the right answers, he poses the right questions.*

-- Claude Levi-Strauss
Why do I get permission errors related to SSLMutex when I start Apache?

Errors such as `mod_ssl: Child could not open SSLMutex lockfile /opt/apache/logs/ssl_mutex.18332 (System error follows) [...] System: Permission denied (errno: 13)` are usually caused by overly restrictive permissions on the parent directories. Make sure that all parent directories (here /opt, /opt/apache and /opt/apache/logs) have the x-bit set for, at minimum, the UID under which Apache's children are running (see the User directive).

Why does mod_ssl stop with the error "Failed to generate temporary 512 bit RSA private key" when I start Apache?

Cryptographic software needs a source of unpredictable data to work correctly. Many open source operating systems provide a "randomness device" that serves this purpose (usually named /dev/random). On other systems, applications have to seed the OpenSSL Pseudo Random Number Generator (PRNG) manually with appropriate data before generating keys or performing public key encryption. As of version 0.9.5, the OpenSSL functions that need randomness report an error if the PRNG has not been seeded with at least 128 bits of randomness.

To prevent this error, mod_ssl has to provide enough entropy to the PRNG to allow it to work correctly. This can be done via the
SSLRandomSeed directive.
Is it possible to provide HTTP and HTTPS from the same server?

Yes. HTTP and HTTPS use different server ports (HTTP binds to port 80, HTTPS to port 443), so there is no direct conflict between them. You can either run two separate server instances bound to these ports, or use Apache's elegant virtual hosting facility to create two virtual servers, both served by the same instance of Apache - one responding over HTTP to requests on port 80, and the other responding over HTTPS to requests on port 443.

Which port does HTTPS use?

You can run HTTPS on any port, but the standards specify port 443, which is where any HTTPS compliant browser will look by default. You can force your browser to look on a different port by specifying it in the URL. For example, if your server is set up to serve pages over HTTPS on port 8080, you can access them at https://example.com:8080/
How do I speak HTTPS manually for testing purposes?

While you usually just use

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

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

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

Before the actual HTTP response you will receive detailed information about the SSL handshake. For a more general command line client which directly understands both HTTP and HTTPS, can perform GET and POST operations, can use a proxy, supports byte ranges, etc. you should have a look at the nifty `cURL` tool. Using this, you can check that Apache is responding correctly to requests via HTTP and HTTPS as follows:

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

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

This can happen when you try to connect to a HTTPS server (or virtual server) via HTTP (eg, using `http://example.com/` instead of `https://example.com`). It can also happen when trying to connect via HTTPS to a HTTP server (eg, using `https://example.com/` on a server which doesn't support HTTPS, or which supports it on a non-standard port). Make sure
that you're connecting to a (virtual) server that supports SSL.

**Why do I get ``Connection Refused'' messages, when trying to access my newly installed Apache+mod_ssl server via HTTPS?**

This error can be caused by an incorrect configuration. Please make sure that your `Listen` directives match your `<VirtualHost>` directives. If all else fails, please start afresh, using the default configuration provided by `mod_ssl`.

**Why are the SSL_XXX variables not available to my CGI & SSI scripts?**

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

**How can I switch between HTTP and HTTPS in relative hyperlinks?**

Usually, to switch between HTTP and HTTPS, you have to use fully-qualified hyperlinks (because you have to change the URL scheme). Using `mod_rewrite` however, you can manipulate relative hyperlinks, to achieve the same effect.

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

This rewrite ruleset lets you use hyperlinks of the form `<a href="document.html_SSL">`, to switch to HTTPS in a relative link. (Replace SSL with NOSSL to switch to HTTP.)
What are RSA Private Keys, CSRs and Certificates?
An RSA private key file is a digital file that you can use to decrypt messages sent to you. It has a public component which you distribute (via your Certificate file) which allows people to encrypt those messages to you.

A Certificate Signing Request (CSR) is a digital file which contains your public key and your name. You send the CSR to a Certifying Authority (CA), who will convert it into a real Certificate, by signing it.

A Certificate contains your RSA public key, your name, the name of the CA, and is digitally signed by the CA. Browsers that know the CA can verify the signature on that Certificate, thereby obtaining your RSA public key. That enables them to send messages which only you can decrypt.

See the Introduction chapter for a general description of the SSL
Is there a difference on startup between a non-SSL-aware Apache and an SSL-aware Apache?

Yes. In general, starting Apache with `mod_ssl` built-in is just like starting Apache without it. However, if you have a passphrase on your SSL private key file, a startup dialog will pop up which asks you to enter the pass phrase.

Having to manually enter the pass phrase when starting the server can be problematic - for example, when starting the server from the system boot scripts. In this case, you can follow the steps below to remove the pass phrase from your private key. Bear in mind that doing so brings additional security risks - proceed with caution!

How do I create a self-signed SSL Certificate for testing purposes?

1. Make sure OpenSSL is installed and in your PATH.

2. Run the following command, to create server.key and server.crt files:
   $ openssl req -new -x509 -nodes -out server.crt -keyout server.key
   These can be used as follows in your `httpd.conf` file:
   ```
   SSLCertificateFile   "/path/to/this/server.crt"
   SSLCertificateKeyFile "/path/to/this/server.key"
   ```

3. It is important that you are aware that this server.key does not have any pass phrase. To add a pass phrase to the key,
you should run the following command, and enter & verify the passphrase as requested.
$ openssl rsa -des3 -in server.key -out server.key.new
$ mv server.key.new server.key

Please backup the server.key file, and the passphrase you entered, in a secure location.

**How do I create a real SSL Certificate?**

Here is a step-by-step description:

1. Make sure OpenSSL is installed and in your PATH.

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

   $ openssl genrsa -des3 -out server.key 2048

   Please backup this server.key file and the pass-phrase you entered in a secure location. You can see the details of this RSA private key by using the command:

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

   If necessary, you can also create a decrypted PEM version (not recommended) of this RSA private key with:

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

3. Create a Certificate Signing Request (CSR) with the server RSA private key (output will be PEM formatted):
$ openssl req -new -key server.key -out server.csr

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

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

4. You now have to send this Certificate Signing Request (CSR) to a Certifying Authority (CA) to be signed. Once the CSR has been signed, you will have a real Certificate, which can be used by Apache. You can have a CSR signed by a commercial CA, or you can create your own CA to sign it. Commercial CAs usually ask you to post the CSR into a web form, pay for the signing, and then send a signed Certificate, which you can store in a server.crt file.

   For details on how to create your own CA, and use this to sign a CSR, see below.

   Once your CSR has been signed, you can see the details of the Certificate as follows:

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

5. You should now have two files: server.key and server.crt. These can be used as follows in your httpd.conf file:

   SSLCertificateFile "/path/to/this/server.crt"
   SSLCertificateKeyFile "/path/to/this/server.key"
The server\.csr file is no longer needed.

**How do I create and use my own Certificate Authority (CA)?**

The short answer is to use the CA\.sh or CA\.pl script provided by OpenSSL. Unless you have a good reason not to, you should use these for preference. If you cannot, you can create a self-signed certificate as follows:

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

   
   
   ```
   $ openssl genrsa -des3 -out server\.key 2048
   ```
   
   Please backup this server\.key file and the pass-phrase you entered in a secure location. You can see the details of this RSA private key by using the command:

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

   If necessary, you can also create a decrypted PEM version (not recommended) of this RSA private key with:

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

2. Create a self-signed certificate (X509 structure) with the RSA key you just created (output will be PEM formatted):

   ```
   $ openssl req -new -x509 -nodes -sha1 -days 365 -key server\.key -out server\.crt
   ```
extensions usr_cert

This signs the server CSR and results in a server.crt file. You can see the details of this Certificate using:

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

How can I change the pass-phrase on my private key file?

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

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

The first time you're asked for a PEM pass-phrase, you should enter the old pass-phrase. After that, you'll be asked again to enter a pass-phrase - this time, use the new pass-phrase. If you are asked to verify the pass-phrase, you'll need to enter the new pass-phrase a second time.

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

The reason this dialog pops up at startup and every re-start is that the RSA private key inside your server.key file is stored in encrypted format for security reasons. The pass-phrase is needed to decrypt this file, so it can be read and parsed. Removing the pass-phrase removes a layer of security from your server - proceed with caution!
1. Remove the encryption from the RSA private key (while keeping a backup copy of the original file):

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

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

   $ chmod 400 server.key

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

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

**How do I verify that a private key matches its Certificate?**

A private key contains a series of numbers. Two of these numbers form the "public key", the others are part of the "private key". The "public key" bits are included when you generate a CSR, and subsequently form part of the associated Certificate.

To check that the public key in your Certificate matches the public portion of your private key, you simply need to compare these
numbers. To view the Certificate and the key run the commands:

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

The `modulus' and the `public exponent' portions in the key and the Certificate must match. As the public exponent is usually 65537 and it's difficult to visually check that the long modulus numbers are the same, you can use the following approach:

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

This leaves you with two rather shorter numbers to compare. It is, in theory, possible that these numbers may be the same, without the modulus numbers being the same, but the chances of this are overwhelmingly remote.

Should you wish to check to which key or certificate a particular CSR belongs you can perform the same calculation on the CSR as follows:

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

**How can I convert a certificate from PEM to DER format?**

The default certificate format for OpenSSL is PEM, which is simply Base64 encoded DER, with header and footer lines. For some applications (e.g. Microsoft Internet Explorer) you need the certificate in plain DER format. You can convert a PEM file `cert.pem` into the corresponding DER file `cert.der` using the following command: `$ openssl x509 -in cert.pem -out...`
Why do browsers complain that they cannot verify my server certificate?

One reason this might happen is because your server certificate is signed by an intermediate CA. Various CAs, such as Verisign or Thawte, have started signing certificates not with their root certificate but with intermediate certificates.

Intermediate CA certificates lie between the root CA certificate (which is installed in the browsers) and the server certificate (which you installed on the server). In order for the browser to be able to traverse and verify the trust chain from the server certificate to the root certificate it needs to be given the intermediate certificates. The CAs should be able to provide you such intermediate certificate packages that can be installed on the server.

You need to include those intermediate certificates with the SSLCertificateChainFile directive.
Why do I get lots of random SSL protocol errors under heavy server load?
Why does my webserver have a higher load, now that it serves SSL encrypted traffic?
Why do HTTPS connections to my server sometimes take up to 30 seconds to establish a connection?
What SSL Ciphers are supported by mod_ssl?
Why do I get `no shared cipher` errors, when trying to use Anonymous Diffie-Hellman (ADH) ciphers?
Why do I get a 'no shared ciphers' error when connecting to my newly installed server?
Why can't I use SSL with name-based/non-IP-based virtual hosts?
Is it possible to use Name-Based Virtual Hosting to identify different SSL virtual hosts?
How do I get SSL compression working?
When I use Basic Authentication over HTTPS the lock icon in Netscape browsers stays unlocked when the dialog pops up. Does this mean the username/password is being sent unencrypted?
Why do I get I/O errors when connecting via HTTPS to an Apache+mod_ssl server with Microsoft Internet Explorer (MSIE)?
How do I enable TLS-SRP?
Why do I get handshake failures with Java-based clients when using a certificate with more than 1024 bits?

Why do I get lots of random SSL protocol errors under heavy server load?

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

**Why does my webserver have a higher load, now that it serves SSL encrypted traffic?**

SSL uses strong cryptographic encryption, which necessitates a lot of number crunching. When you request a webpage via HTTPS, everything (even the images) is encrypted before it is transferred. So increased HTTPS traffic leads to load increases.

**Why do HTTPS connections to my server sometimes take up to 30 seconds to establish a connection?**

This is usually caused by a /dev/random device for SSLRandomSeed which blocks the read(2) call until enough entropy is available to service the request. More information is available in the reference manual for the SSLRandomSeed directive.

**What SSL Ciphers are supported by mod_ssl?**

Usually, any SSL ciphers supported by the version of OpenSSL in use, are also supported by mod_ssl. Which ciphers are available can depend on the way you built OpenSSL. Typically, at least the following ciphers are supported:

1. RC4 with SHA1
2. AES with SHA1
3. Triple-DES with SHA1

To determine the actual list of ciphers available, you should run the following:

```
$ openssl ciphers -v
```
Why do I get "no shared cipher" errors, when trying to use Anonymous Diffie-Hellman (ADH) ciphers?

By default, OpenSSL does not allow ADH ciphers, for security reasons. Please be sure you are aware of the potential side-effects if you choose to enable these ciphers.

In order to use Anonymous Diffie-Hellman (ADH) ciphers, you must build OpenSSL with ``-DSSL_ALLOW_ADH'', and then add ``ADH'' into your SSLCipherSuite.

Why do I get a 'no shared ciphers' error when connecting to my newly installed server?

Either you have made a mistake with your SSLCipherSuite directive (compare it with the pre-configured example in extra/httpd-ssl.conf) or you chose to use DSA/DH algorithms instead of RSA when you generated your private key and ignored or overlooked the warnings. If you have chosen DSA/DH, then your server cannot communicate using RSA-based SSL ciphers (at least until you configure an additional RSA-based certificate/key pair). Modern browsers like NS or IE can only communicate over SSL using RSA ciphers. The result is the "no shared ciphers" error. To fix this, regenerate your server certificate/key pair, using the RSA algorithm.

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

The reason is very technical, and a somewhat "chicken and egg" problem. The SSL protocol layer stays below the HTTP protocol layer and encapsulates HTTP. When an SSL connection (HTTPS) is established Apache/mod_ssl has to negotiate the SSL protocol parameters with the client. For this, mod_ssl has to consult the configuration of the virtual server (for instance it has to look for the cipher suite, the server certificate, etc.). But in order to go to the
correct virtual server Apache has to know the Host HTTP header field. To do this, the HTTP request header has to be read. This cannot be done before the SSL handshake is finished, but the information is needed in order to complete the SSL handshake phase. See the next question for how to circumvent this issue.

Note that if you have a wildcard SSL certificate, or a certificate that has multiple hostnames on it using subjectAltName fields, you can use SSL on name-based virtual hosts without further workarounds.

**Is it possible to use Name-Based Virtual Hosting to identify different SSL virtual hosts?**

Name-Based Virtual Hosting is a very popular method of identifying different virtual hosts. It allows you to use the same IP address and the same port number for many different sites. When people move on to SSL, it seems natural to assume that the same method can be used to have lots of different SSL virtual hosts on the same server.

It is possible, but only if using a 2.2.12 or later web server, built with 0.9.8j or later OpenSSL. This is because it requires a feature that only the most recent revisions of the SSL specification added, called Server Name Indication (SNI).

Note that if you have a wildcard SSL certificate, or a certificate that has multiple hostnames on it using subjectAltName fields, you can use SSL on name-based virtual hosts without further workarounds.

The reason is that the SSL protocol is a separate layer which encapsulates the HTTP protocol. So the SSL session is a separate transaction, that takes place before the HTTP session has begun. The server receives an SSL request on IP address X and port Y (usually 443). Since the SSL request did not contain any Host: field, the server had no way to decide which SSL virtual host to
use. Usually, it just used the first one it found which matched the port and IP address specified.

If you are using a version of the web server and OpenSSL that support SNI, though, and the client’s browser also supports SNI, then the hostname is included in the original SSL request, and the web server can select the correct SSL virtual host.

You can, of course, use Name-Based Virtual Hosting to identify many non-SSL virtual hosts (all on port 80, for example) and then have a single SSL virtual host (on port 443). But if you do this, you must make sure to put the non-SSL port number on the NameVirtualHost directive, e.g.

```
NameVirtualHost 192.168.1.1:80
```

Other workaround solutions include:

Using separate IP addresses for different SSL hosts. Using different port numbers for different SSL hosts.

**How do I get SSL compression working?**

Although SSL compression negotiation was defined in the specification of SSLv2 and TLS, it took until May 2004 for RFC 3749 to define DEFLATE as a negotiable standard compression method.

OpenSSL 0.9.8 started to support this by default when compiled with the zlib option. If both the client and the server support compression, it will be used. However, most clients still try to initially connect with an SSLv2 Hello. As SSLv2 did not include an array of preferred compression algorithms in its handshake, compression cannot be negotiated with these clients. If the client disables support for SSLv2, either an SSLv3 or TLS Hello may be
sent, depending on which SSL library is used, and compression may be set up. You can verify whether clients make use of SSL compression by logging the \%{SSL_COMPRESS_METHOD}\x variable.

**When I use Basic Authentication over HTTPS the lock icon in Netscape browsers stays unlocked when the dialog pops up. Does this mean the username/password is being sent unencrypted?**

No, the username/password is transmitted encrypted. The icon in Netscape browsers is not actually synchronized with the SSL/TLS layer. It only toggles to the locked state when the first part of the actual webpage data is transferred, which may confuse people. The Basic Authentication facility is part of the HTTP layer, which is above the SSL/TLS layer in HTTPS. Before any HTTP data communication takes place in HTTPS, the SSL/TLS layer has already completed its handshake phase, and switched to encrypted communication. So don't be confused by this icon.

**Why do I get I/O errors when connecting via HTTPS to an Apache+mod_ssl server with older versions of Microsoft Internet Explorer (MSIE)?**

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

```bash
SetEnvIf User-Agent "MSIE [2-5]"
```
Further, some MSIE versions have problems with particular ciphers. Unfortunately, it is not possible to implement a MSIE-specific workaround for this, because the ciphers are needed as early as the SSL handshake phase. So a MSIE-specific SetEnvIf won't solve these problems. Instead, you will have to make more drastic adjustments to the global parameters. Before you decide to do this, make sure your clients really have problems. If not, do not make these changes - they will affect all your clients, MSIE or otherwise.

How do I enable TLS-SRP?

TLS-SRP (Secure Remote Password key exchange for TLS, specified in RFC 5054) can supplement or replace certificates in authenticating an SSL connection. To use TLS-SRP, set the SSLSRPVerifierFile directive to point to an OpenSSL SRP verifier file. To create the verifier file, use the openssl tool:

```bash
openssl srp -srpvfile passwd.srpv -add username
```

After creating this file, specify it in the SSL server configuration:

```bash
SSLSRPVerifierFile /path/to/passwd.srpv
```

To force clients to use non-certificate TLS-SRP cipher suites, use the following directive:

```bash
SSLCipherSuite "!DSS:!aRSA:SRP"
```

Why do I get handshake failures with Java-based
clients when using a certificate with more than 1024 bits?

Beginning with version 2.4.7, mod_ssl will use DH parameters which include primes with lengths of more than 1024 bits. Java 7 and earlier limit their support for DH prime sizes to a maximum of 1024 bits, however.

If your Java-based client aborts with exceptions such as java.lang.RuntimeException: Could not generate DH keypair and java.security.InvalidAlgorithmParameterException: Prime size must be multiple of 64, and can only range from 512 to 1024 (inclusive), and httpd logs tlsv1 alert internal error (SSL alert number 80) (at LogLevel info or higher), you can either rearrange mod_ssl's cipher list with SSLCipherSuite (possibly in conjunction with SSLHonorCipherOrder), or you can use custom DH parameters with a 1024-bit prime, which will always have precedence over any of the built-in DH parameters.

To generate custom DH parameters, use the openssl dhparam 1024 command. Alternatively, you can use the following standard 1024-bit DH parameters from RFC 2409, section 6.2:

```
-----BEGIN DH PARAMETERS-----
MIGHAoGBAP///////////yQ/aoiFowjTExmKLgNwc0SkCTgiKZ8x0Agu+pjsTmyJR
Sgh5jjQE3e+VGbPN0kMbMCsKbfJfFDdP4TVtbVHCRESftXZiXn7G9ExC6aY37WsL
/1y29Aa37e44a/taiZ+1rp8kEXxLH+ZJKGZR7OZTgf/ ///////////AgEC
-----END DH PARAMETERS-----
```

Add the custom parameters including the "BEGIN DH PARAMETERS" and "END DH PARAMETERS" lines to the end of the first certificate file you have configured using the SSLCertificateFile directive.
- **What information resources are available in case of mod_ssl problems?**
- **What support contacts are available in case of mod_ssl problems?**
- **What information should I provide when writing a bug report?**
- **I had a core dump, can you help me?**
- **How do I get a backtrace, to help find the reason for my core dump?**

### What information resources are available in case of mod_ssl problems?

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

**Answers in the User Manual's F.A.Q. List (this)**

[http://httpd.apache.org/docs/2.4/ssl/sslfaq.html](http://httpd.apache.org/docs/2.4/ssl/sslfaq.html)

First check the F.A.Q. (this text). If your problem is a common one, it may have been answered several times before, and been included in this doc.

### What support contacts are available in case of mod_ssl problems?

The following lists all support possibilities for mod_ssl, in order of preference. Please go through these possibilities *in this order* - don't just pick the one you like the look of.

1. **Send a Problem Report to the Apache httpd Users Support Mailing List**
   
   [users@httpd.apache.org](mailto:users@httpd.apache.org)

   This is the second way of submitting your problem report. Again, you must subscribe to the list first, but you can then easily discuss your problem with the whole Apache httpd user
2. **Write a Problem Report in the Bug Database**

   This is the last way of submitting your problem report. You should only do this if you've already posted to the mailing lists, and had no success. Please follow the instructions on the above page *carefully*.

**What information should I provide when writing a bug report?**

You should always provide at least the following information:

**Apache httpd and OpenSSL version information**

The Apache version can be determined by running `httpd -v`. The OpenSSL version can be determined by running `openssl version`. Alternatively, if you have Lynx installed, you can run the command `lynx -mime_header http://localhost/ | grep Server` to gather this information in a single step.

**The details on how you built and installed Apache httpd and OpenSSL**

For this you can provide a logfile of your terminal session which shows the configuration and install steps. If this is not possible, you should at least provide the `configure` command line you used.

**In case of core dumps please include a Backtrace**

If your Apache httpd dumps its core, please attach a stack-frame ``backtrace'' (see [below](#) for information on how to get this). This information is required in order to find a reason for your core dump.

**A detailed description of your problem**

Don't laugh, we really mean it! Many problem reports don't
include a description of what the actual problem is. Without this, it's very difficult for anyone to help you. So, it's in your own interest (you want the problem be solved, don't you?) to include as much detail as possible, please. Of course, you should still include all the essentials above too.

I had a core dump, can you help me?

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

How do I get a backtrace, to help find the reason for my core dump?

Following are the steps you will need to complete, to get a backtrace:

1. Make sure you have debugging symbols available, at least in Apache. On platforms where you use GCC/GDB, you will have to build Apache+mod_ssl with ``OPTIM="-g -ggdb3"'' to get this. On other platforms at least ``OPTIM="-g"'' is needed.

2. Start the server and try to reproduce the core-dump. For this you may want to use a directive like ``CoreDumpDirectory /tmp'' to make sure that the core-dump file can be written. This should result in a /tmp/core or /tmp/httpd.core file. If you don't get one of these, try running your server under a non-root UID. Many modern kernels do not allow a process to dump core after it has done a setuid() (unless it does an exec()) for security reasons (there can be privileged information left over in memory). If necessary, you can run
/path/to/httpd -X manually to force Apache to not fork.

3. Analyze the core-dump. For this, run `gdb /path/to/httpd /tmp/httpd.core` or a similar command. In GDB, all you have to do then is to enter `bt`, and voila, you get the backtrace. For other debuggers consult your local debugger manual.
3

- ( **AuthType** )
  - mod_auth_basic
  - mod_auth_digest

- ( **AuthBasicProvider**, **AuthDigestProvider** )
  - mod_authn_anon
  - mod_authn_dbd
  - mod_authn_dbm
  - mod_authn_default
  - mod_authn_file
  - mod_authnz_ldap

- ( **Require** )
  - mod_authnz_ldap
  - mod_authz_dbm
  - mod_authz_dbm
  - mod_authz_default
  - mod_authz_groupfile
  - mod_authz_host
  - mod_authz_owner
  - mod_authz_user

  mod_authn_core  mod_authz_core  2

  mod_authnz_ldap
  mod_access
  mod_access_comp
mod_ssl
<Directory .htaccess>
  AllowOverride AuthConfig
</Directory>

mod_authn_core  mod_authz_core  httpd  httpd.conf
httpd

*
htpasswd -c /usr/local/apache/passwd/passwords rbowen

tpasswd

# htpasswd -c /usr/local/apache/passwd/passwords rbowen
New password: mypassword
Re-type new password: mypassword
Adding password for user rbowen

htpasswd

/usr/local/apache/bin/htpasswd

/usr/local/apache/htdocs/secret
/usr/local/apache/htdocs/secret/.htaccess
httpd.conf <Directory /usr/local/apache/htdocs/secret>

AuthType Basic
AuthName "Restricted Files"
# (Following line optional)
AuthBasicProvider file
AuthUserFile /usr/local/apache/passwd/passwords
Require user rbowen
AuthType
Basic
AuthType Digest
mod_auth_digest
AuthName
Realm (: )
Realm
"Restricted Files"
AuthBasicProvider
file
mod_authn_dbd
AuthUserFile
htpasswd
mod_authn_dbm
AuthDBMUserFile
Apache
Require
GroupName: rbowen dpitts sungo rshersey

htpasswd /usr/local/apache/passwd/passwords dpitts

.htaccess

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

GroupName password

Require valid-user

require user rbowen
Basic
<Directory /www/docs/private>
  AuthName "Private"
  AuthType Basic
  AuthBasicProvider dbm
  AuthDBMUserFile /www/passwords/passwd.dbm
  Require valid-user
</Directory>
<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>

<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>

<SatisfyAll>  <SatisfyOne>  A^
**AND/OR**

**Require**

```plaintext
<SatisfyAll>, <SatisfyOne>

# if ((user == "John") ||
#     ((Group == "admin")
#     && (ldap-group <ldap-object> contains auth'ed_user)
#     && ((ldap-attribute dept == "sales")
#         || (file-group contains auth'ed_user))))
# then
#     auth_granted
# else
#     auth_denied
#
<Directory /www/mydocs>
    Authname ...
    AuthBasicProvider ...  ...
    Require user John
    <SatisfyAll>
        Require Group admins
        Require ldap-group cn=mygroup,o=foo
        <SatisfyOne>
            Require ldap-attribute dept="sales"
            Require file-group
        </SatisfyOne>
    </SatisfyAll>
</Directory>
```

**Require**  **OR**

**AND**

**Require  Reject**
```
all, env, host, ip  IP

Require  Reject

Require ip address

address  IP ( IP ) :

Require host domain_name

domain_name  FQDN () :

Reject ip 10.252.46.165

Reject host host.example.com

IP :

<SatisfyAll>
  Reject ip 192.168.205
  Reject host phishers.example.com moreidiots.example
  Reject host ke
</SatisfyAll>

Reject  <SatisfyAll>

<SatisfyAll>  Reject

 mod access compat

Order, !
```
mod_authz_default
mod_authz_default
<AuthnProviderAlias>
mod_alias  AddHandler
mod_cgi    Options
          ScriptAlias

CGI (Common Gateway Interface)
### ScriptAlias

<table>
<thead>
<tr>
<th>Apache</th>
<th>ht</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Alias</strong></td>
<td><strong>ScriptAlias</strong></td>
</tr>
<tr>
<td>/cgi-bin/</td>
<td>/usr/local/apache2/cgi-bin/</td>
</tr>
</tbody>
</table>

### URL

- http://www.example.com/cgi-bin/test.pl
- /usr/local/apache2/cgi-bin/test.pl

### Options

- ExecCGI
CGI Options

Options

<Directory /usr/local/apache2/htdocs/somedir>
  Options +ExecCGI
</Directory>

CGI

AddHandler cgi-script .cgi .pl

.htaccess

.httpd.conf

User

.cgi

<Directory /home/*/public_html>
  Options +ExecCGI
  AddHandler cgi-script .cgi
</Directory>

cgi-bin

<Directory /home/*/public_html/cgi-bin>
  Options ExecCGI
  SetHandler cgi-script
</Directory>
Content-type: text/html

Hello, World.
CGI

"POST Method Not Allowed"

CGI Apache

"Forbidden"

"Internal Server Error"

Apache "Premature end of script headers"

CGI HTTP

```
chmod a+x first.pl
```

( perl) CGI 1:

```
#!/usr/bin/perl
```
CGI

Apache

CGI

CGI

```
cd /usr/local/apache2/cgi-bin
./first.pl

(perl

Content-Type HTTP

end of script headers

Suexec

suexec script headers

suexec apachectl -V SUEXEC_BIN

suexec

suexec

suexec

suexec

suexec

apachectl -V suexec
```
#!/usr/bin/perl
print "Content-type: text/html\n\n";
foreach $key (keys %ENV) {
    print "$key --> $ENV{$key}<br>";
}

name=Rich%20Bowen&city=Lexington&state=KY&sidekick=Squirrel%20Monkey
CGI

Perl  CGI  CPAN

C  CGI
CGI

HTML Writers Guide
http://www.hwg.org/lists/hwg-servers/

CGI

Apache

CGI
Apache: Server Side Includes

HTML
| mod include | Options |
| modcgi    | XBitHack |
| mod expires | AddType |
|           | SetOutputFilter |
|           | BrowserMatchNoCase |

SSI  Server Side Includes
SSI

SSI  SSI

SSI  SSI
SSI (Server Side Includes) HTML

SSI
SSI  httpd.conf  .htaccess:

Options +Includes

SSI  Apache Options

AddType text/html .shtml
AddOutputFilter INCLUDES .shtml

SSI

XBitHack:

XBitHack on

XBitHack chmod

chmod +x pagename.html

.html SSI Apache

XBitHack

Windows

Apache SSI

1. XBitHack Full
2. mod_expires

http://example.com
SSI:

<!--#element attribute=value attribute=value ... -->

HTML SSI

element

<!--#echo var="DATE_LOCAL" -->

echo

CGI

config
timefmt

<!--#config timefmt="%A %B %d, %Y" -->

Today is <!--#echo var="DATE_LOCAL" -->

This document last modified <!--#lastmod file="index.html" -->

timefmt

CGI

````

<!--#include virtual="/cgi-bin/counter.pl" -->
config

SSI

[an error occurred while processing this directive]

config  errmsg:

<!--#config errmsg="[It appears that you don't know how to use SSI]" -->

config  sizefmt

abbrev
CGI  SSI
Win32  DOS)

<pre>
<!--#exec cmd="ls" -->
</pre>

Windows

<pre>
<!--#exec cmd="dir" -->
</pre>

Windows

exec
IncludesNOEXEC  SSI  exec
Apache SSI

Apache 1.2

set

<!--#set var="name" value="Rich" -->

(  LAST_MODIFIED

<!--#set var="modified" value="$LAST_MODIFIED" -->

($)  

<!--#set var="cost" value="$100" -->

<!--#set var="date" value="$\{DATE_LOCAL\}_\{DATE_GMT\}" -->

endif:

<!--#if expr="test_condition" -->
<!--#elif expr="test_condition" -->
<!--#else -->
<!--#endif -->
test_condition

:

BrowserMatchNoCase macintosh Mac
BrowserMatchNoCase MSIE InternetExplorer

Macintosh

SSI:

<!--[if expr="${Mac} && ${InternetExplorer}"]-->
Apologetic text goes here
<!--[endif]-->

<!--[if !expr="${Mac} && ${InternetExplorer}"]-->
Cool JavaScript code goes here
<!--[endif]-->

Mac IE

JavaScript Mac

( )
Apache : .htaccess

.htaccess

▶
<table>
<thead>
<tr>
<th>core</th>
<th>AccessFileName</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_authn_file</td>
<td>AllowOverride</td>
</tr>
<tr>
<td>mod_authz_groupfile</td>
<td>Options</td>
</tr>
<tr>
<td>mod_cgi</td>
<td>AuthType</td>
</tr>
<tr>
<td>mod_include</td>
<td>AuthName</td>
</tr>
<tr>
<td>mod_mime</td>
<td>AuthUserFile</td>
</tr>
<tr>
<td></td>
<td>AuthGroupFile</td>
</tr>
<tr>
<td></td>
<td>Require</td>
</tr>
</tbody>
</table>
.htaccess()

: .htaccess

  AccessFileName .config

.htaccess

  AllowOverride

  AddDefaultCharset

  FileInfo

  FileInfo

: .htaccess

  "..." .htaccess

  FileInfo

  FileInfo

  ".htaccess"
.htaccess

ISP

.htaccess

AllowOverride .htaccess Apache

.htaccess

Apache

/www/htdocs_example

/.htaccess
/www/.htaccess
/www/htdocs/.htaccess
/www/htdocs_example/.htaccess

/www/htdocs_example .htaccess

<Directory /www/htdocs_example>

AddType text/example .exm

/www/htdocs_example .htaccess
### httpd.conf file

```bash
<Directory /www/htdocs/example>
  AddType text/example .exm
</Directory>
```

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AllowOverride</td>
<td>none .htaccess</td>
</tr>
</tbody>
</table>

- AllowOverride None
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:

```
.htaccess
<Directory>
AllowOverride
```
<Directory />  
   Allowoverride All  
</Directory>

<Location />  
   Options +IncludesNoExec -ExecCGI  
</Location>
.htaccess

AuthType Basic
AuthName "Password Required"
AuthUserFile /www/passwords/password.file
AuthGroupFile /www/passwords/group.file
Require Group admins

AllowOverride AuthConfig
### .htaccess

Options +Includes
AddType text/html shtml
AddHandler server-parsed shtml

### AllowOverride Options
CGI

Options +ExecCGI
AddHandler cgi-script cgi pl

CGI

Options +ExecCGI
SetHandler cgi-script

AllowOverride Options
.htaccess

Apache

AllowOverride .htaccess

Apache
UserDir "username" UserDir

in the default config file, and adapting the httpd-userdir.conf file as necessary, or by including the appropriate directives in a Directory block within the main config file.

UserDir

#include conf/extra/httpd-userdir.conf

Directory

URL
<table>
<thead>
<tr>
<th>mod_userdir</th>
<th>UserDir</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectoryMatch</td>
<td>AllowOverride</td>
</tr>
</tbody>
</table>
UserDir

UserDir public_html

URL http://example.com/~rbowen/file.html
/home/rbowen/public_html/file.html

UserDir /var/html

URL http://example.com/~rbowen/file.html
/var/html/rbowen/file.html

(*)

UserDir /var/www/*/docs

URL http://example.com/~rbowen/file.html
/var/www/rbowen/docs/file.html

UserDir public_html /var/html

http://example.com/~rbowen/file.html   URL
~rbowen /var/html rbowen
/var/html/rbowen/file.html
UserDir  URL

UserDir http://example.org/users/*/ 

http://example.com/~bob/abc.html
http://example.org/users/bob/abc.html
<table>
<thead>
<tr>
<th>UserDir enabled</th>
<th>UserDir disabled</th>
<th>root</th>
<th>jro</th>
<th>fish</th>
</tr>
</thead>
<tbody>
<tr>
<td>UserDir disabled</td>
<td>UserDir enabled</td>
<td>rbowen</td>
<td>krietz</td>
<td></td>
</tr>
</tbody>
</table>
<Directory /home/*/public_html/cgi-bin/>
  Options ExecCGI
  SetHandler cgi-script
</Directory>

UserDir  public_html

http://example.com/~rbowen/cgi-bin/example.cgi
Using Apache HTTP Server on Microsoft Windows

This document explains how to install, configure and run Apache 2.4 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.
The primary Windows platform for running Apache 2.4 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.
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
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:

```plaintext
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 Apache2.4 is installed at C:\Program Files,
then the root directory, Program Files and Apache2.4 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 \Apache2.4\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.
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 Apache2.4 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 Apache2.4` and `NET STOP Apache2.4` 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 Apache2.4 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 Apache2.4 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 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 2.4.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\Apache2.4\`, you can do the following:

```
c: cd "\Program Files\Apache Software Foundation\Apache2.4\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.
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.
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**

```bash
DocumentRoot "//dochost/www/html/"
```

**Example DocumentRoot with IP address in UNC path**

```bash
DocumentRoot "/192.168.1.50/docs/"
```

**Example Alias and corresponding Directory with UNC path**

```bash
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.
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.
Compiling Apache for Microsoft Windows

There are many important points to consider before you begin compiling Apache HTTP Server (httpd). See Using Apache HTTP Server on Microsoft Windows before you begin.

httpd can be built on Windows using a cmake-based build system or with Visual Studio project files maintained by httpd developers. The cmake-based build system directly supports more versions of Visual Studio but currently has considerable functional limitations.
Requirements

Compiling Apache requires the following environment to be properly installed:

- **Disk Space**

  Make sure you have at least 200 MB of free disk space available. After installation Apache requires approximately 80 MB of disk space, plus space for log and cache files, which can grow rapidly. The actual disk space requirements will vary considerably based on your chosen configuration and any third-party modules or libraries, especially when OpenSSL is also built. Because many files are text and very easily compressed, NTFS filesystem compression cuts these requirements in half.

- **Appropriate Patches**

  The httpd binary is built with the help of several patches to third party packages, which ensure the released code is buildable and debuggable. These patches are available and distributed from http://www.apache.org/dist/httpd/binaries/win32/patches_applied and are recommended to be applied to obtain identical results as the "official" ASF distributed binaries.

- **Microsoft Visual C++ 6.0 (Visual Studio 97) or later.**

  Apache can be built using the command line tools, or from within the Visual Studio IDE Workbench. The command line build requires the environment to reflect the PATH, INCLUDE, LIB and other variables that can be configured with the
You may want the Visual Studio Processor Pack for your older version of Visual Studio, or a full (not Express) version of newer Visual Studio editions, for the ml.exe assembler. This will allow you to build OpenSSL, if desired, using the more efficient assembly code implementation.

Only the Microsoft compiler tool chain is actively supported by the active httpd contributors. Although the project regularly accepts patches to ensure MinGW and other alternative builds work and improve upon them, they are not actively maintained and are often broken in the course of normal development.

- Updated Microsoft Windows Platform SDK, February 2003 or later.

An appropriate Windows Platform SDK is included by default in the full (not express/lite) versions of Visual C++ 7.1 (Visual Studio 2002) and later, these users can ignore these steps unless explicitly choosing a newer or different version of the Platform SDK.

To use Visual C++ 6.0 or 7.0 (Studio 2000 .NET), the Platform SDK environment must be prepared using the setenv.bat script (installed by the Platform SDK) before starting the command line build or launching the msdev/devenv GUI environment. Installing the Platform SDK for Visual Studio Express versions (2003 and later) should adjust the default environment appropriately.

"c:\Program Files\Microsoft Visual Studio\VC98\Bin\VCVARS32"
Perl and awk

Several steps recommended here require a perl interpreter during the build preparation process, but it is otherwise not required.

To install Apache within the build system, several files are modified using the awk.exe utility. awk was chosen since it is a very small download (compared with Perl or WSH/VB) and accomplishes the task of modifying configuration files upon installation. Brian Kernighan's [http://www.cs.princeton.edu/~bwk/btl.mirror/](http://www.cs.princeton.edu/~bwk/btl.mirror/) site has a compiled native Win32 binary, [http://www.cs.princeton.edu/~bwk/btl.mirror/awk95.exe](http://www.cs.princeton.edu/~bwk/btl.mirror/awk95.exe) which you must save with the name awk.exe (rather than awk95.exe).

If awk.exe is not found, Makefile.win's install target will not perform substitutions in the installed .conf files. You must manually modify the installed .conf files to allow the server to start. Search and replace all "@token@" tags as appropriate.

The Visual Studio IDE will only find awk.exe from the PATH, or executable path specified in the menu option Tools -> Options -> (Projects ->) Directories. Ensure awk.exe is in your system path.

Also note that if you are using Cygwin tools ([http://www.cygwin.com/](http://www.cygwin.com/)) the awk utility is named gawk.exe and that the file awk.exe is really a symlink to
the gawk.exe file. The Windows command shell does not recognize symlinks, and because of this building InstallBin will fail. A workaround is to delete awk.exe from the cygwin installation and copy gawk.exe to awk.exe. Also note the cygwin/mingw ports of gawk 3.0.x were buggy, please upgrade to 3.1.x before attempting to use any gawk port.

- [Optional] zlib library (for mod_deflate)

Zlib must be installed into a srclib subdirectory named zlib. This must be built in-place. Zlib can be obtained from http://www.zlib.net/ -- the mod_deflate is confirmed to work correctly with version 1.2.3.

    nmake -f win32\Makefile.msc
    nmake -f win32\Makefile.msc test

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

    The OpenSSL library is cryptographic software. The country in which you currently reside may have restrictions on the import, possession, use, and/or re-export to another country, of encryption software. BEFORE using any encryption software, please check your country's laws, regulations and policies concerning the import, possession, or use, and re-export of encryption software, to see if this is permitted. See http://www.wassenaar.org/ for more information.

    Configuring and building OpenSSL requires perl to be installed.

    OpenSSL must be installed into a srclib subdirectory
named openssl, obtained from
http://www.openssl.org/source/, in order to compile mod_ssl
or the abs.exe project, which is ab.c with SSL support
enabled. To prepare OpenSSL to be linked to Apache
mod_ssl or abs.exe, and disable patent encumbered features
in OpenSSL, you might use the following build commands:

```
perl Configure no-rc5 no-idea enable-mdc2 enable-zlib VC-
WIN32 -Ipath/to/srclib/zlib -Lpath/to/srclib/zlib
ms\do_masm.bat
nmake -f ms\ntdll.mak
```

It is not advisable to use zlib-dynamic, as that transfers the
cost of deflating SSL streams to the first request which
must load the zlib dll. Note the suggested patch enables the
-L flag to work with windows builds, corrects the name of
zdll.lib and ensures .pdb files are generated for
troubleshooting. If the assembler is not installed, you would
add no-asm above and use ms\do_ms.bat instead of the
ms\do_masm.bat script.

- [Optional] Database libraries (for mod_dbd and
  mod_authn_dbm)

The apr-util library exposes dbm (keyed database) and dbd
(query oriented database) client functionality to the httpd
server and its modules, such as authentication and
authorization. The sdbm dbm and odbc dbd providers are
compiled unconditionally.

The dbd support includes the Oracle instantclient package,
MySQL, PostgreSQL and sqlite. To build these all, for
example, set up the LIB to include the library path, INCLUDE
to include the headers path, and PATH to include the dll bin
path of all four SDK’s, and set the DBD_LIST environment
variable to inform the build which client driver SDKs are
installed correctly, e.g.;
set DBD_LIST=sqlite3 pgsql oracle mysql

Similarly, the dbm support can be extended with DBM_LIST to
build a Berkeley DB provider (db) and/or gdbm provider, by
similarly configuring LIB, INCLUDE and PATH first to ensure
the client library libs and headers are available.
set DBM_LIST=db gdbm

Depending on the choice of database distributions, it may
be necessary to change the actual link target name (e.g.
gdbm.lib vs. libgdb.lib) that are listed in the corresponding
.dsp/.mak files within the directories srclib\apr-util\dbd or
...\dbm.
See the README-win32.txt file for more hints on obtaining
the various database driver SDKs.

Building from Unix sources
The policy of the Apache HTTP Server project is to only release
Unix sources. Windows source packages made available for
download have been supplied by volunteers and may not be
available for every release. You can still build the server on
Windows from the Unix source tarball with just a few additional
steps.
1. Download and unpack the Unix source tarball for the latest
version.
2. Download and unpack the Unix source tarball for latest
version of APR, AR-Util and APR-Iconv, place these sources


in directories httpd-2.x.x\src\lib\apr, httpd-2.x.x\src\lib\apr-util and httpd-2.x.x\src\lib\apr-iconv

3. Open a Command Prompt and CD to the httpd-2.x.x folder
4. Run the line endings conversion utility at the prompt;

perl src\lib\apr\build\lineends.pl

You can now build the server with the Visual Studio development environment using the IDE. Command-Line builds of the server are not possible from Unix sources unless you export .mak files as explained below.

**Command-Line Build**

Makefile.win is the top level Apache makefile. To compile Apache on Windows, simply use one of the following commands to build the release or debug flavor:

```
nmake /f Makefile.win _apacher
nmake /f Makefile.win _apached
```

Either command will compile Apache. The latter will disable optimization of the resulting files, making it easier to single step the code to find bugs and track down problems.

You can add your apr-util dbd and dbm provider choices with the additional make (environment) variables DBD_LIST and DBM_LIST, see the comments about [Optional] Database libraries, above. Review the initial comments in Makefile.win for additional options that can be provided when invoking the build.

**Developer Studio Workspace IDE Build**

Apache can also be compiled using VC++'s Visual Studio
development environment. To simplify this process, a Visual Studio workspace, Apache .dsw, is provided. This workspace exposes the entire list of working .dsp projects that are required for the complete Apache binary release. It includes dependencies between the projects to assure that they are built in the appropriate order.

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

The .dsp project files are distributed in Visual Studio 6.0 (98) format. Visual C++ 5.0 (97) will recognize them. Visual Studio 2002 (.NET) and later users must convert Apache .dsw plus the .dsp files into an Apache .sln plus .msproj files. Be sure you reconvert the .msproj file again if its source .dsp file changes! This is really trivial, just open Apache .dsw in the VC++ 7.0 IDE once again and reconvert.

There is a flaw in the .vcproj conversion of .dsp files. devenv.exe will mis-parse the /D flag for RC flags containing long quoted /D 'efines which contain spaces. The command:

```
perl srclib\apr\build\cvtdsp.pl -2005
```

will convert the /D flags for RC flags to use an alternate, parseable syntax; unfortunately this syntax isn't supported by
Visual Studio 97 or its exported .mak files. These /D flags are used to pass the long description of the mod_apachemodule.so files to the shared .rc resource version-identifier build.

**Building with OpenSSL 1.1.0 and up** Due to difference in the build structure of OpenSSL begining with version 1.1.0 you will need to convert the dsp files affected with cvtdsp.pl from APR 1.6 or greater. The command:

```
perl srclib\apr\build\cvtdsp.pl -oss11
```

Visual Studio 2002 (.NET) and later users should also use the Build menu, Configuration Manager dialog to uncheck both the Debug and Release Solution modules abs, **mod_deflate** and **mod_ssl** components, as well as every component starting with apr_db*. These modules are built by invoking nmake, or the IDE directly with the BinBuild target, which builds those modules conditionally if the srclib directories openssl and/or zlib exist, and based on the setting of DBD_LIST and DBM_LIST environment variables.

**Exporting command-line .mak files**

Exported .mak files pose a greater hassle, but they are required for Visual C++ 5.0 users to build **mod_ssl**, abs (**ab** with SSL support) and/or **mod_deflate**. The .mak files also support a broader range of C++ tool chain distributions, such as Visual Studio Express.

You must first build all projects in order to create all dynamic auto-generated targets, so that dependencies can be parsed correctly. Build the entire project from within the Visual Studio 6.0 (98) IDE,
using the BuildAll target, then use the Project Menu Export for all makefiles (checking on "with dependencies"). Run the following command to correct absolute paths into relative paths so they will build anywhere:

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

You must type this command from the top level directory of the httpd source tree. Every .mak and .dep project file within the current directory and below will be corrected, and the timestamps adjusted to reflect the .dsp.

Always review the generated .mak and .dep files for Platform SDK or other local, machine specific file paths. The DevStudio\Common\MSDev98\bin\ (VC6) directory contains a sysincl.dat file, which lists all exceptions. Update this file (including both forward and backslashed paths, such as both sys/time.h and sys\time.h) to ignore such newer dependencies. Including local-install paths in a distributed .mak file will cause the build to fail completely.

If you contribute back a patch that revises project files, we must commit project files in Visual Studio 6.0 format. Changes should be simple, with minimal compilation and linkage flags that can be recognized by all Visual Studio environments.

**Installation**

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

To build and install all the files into the desired folder dir automatically, use one of the following nmake commands:
The `dir` argument to INSTDIR provides the installation directory; it can be omitted if Apache is to be installed into `\Apache22` (of the current drive).

**Warning about building Apache from the development tree**

Note only the `.dsp` files are maintained between release builds. The `.mak` files are NOT regenerated, due to the tremendous waste of reviewer's time. Therefore, you cannot rely on the NMAKE commands above to build revised `.dsp` project files unless you then export all `.mak` files yourself from the project. This is unnecessary if you build from within the Microsoft Developer Studio environment.
The primary documentation for this build mechanism is in the README.cmake file in the source distribution. Refer to that file for detailed instructions.

Building httpd with cmake requires building APR and APR-util separately. Refer to their README.cmake files for instructions.

The primary limitations of the cmake-based build are inherited from the APR-util project, and are listed below because of their impact on httpd:

- No cmake build for the APR-iconv subproject is available, and the APR-util cmake build cannot consume an existing APR-iconv build. Thus, mod_charset_lite and possibly some third-party modules cannot be used.
- The cmake build for the APR-util subproject does not support most of the optional DBM and DBD libraries supported by the included Visual Studio project files. This limits the database backends supported by a number of bundled and third-party modules.
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.
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.
Information on the latest version of Apache can be found on the Apache web server at [http://www.apache.org/](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](http://www.apache.org/).
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).
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:
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.
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:

  ```apache
  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 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](http://developer.novell.com/ndk/apache.htm) or higher.
- [NetWare Libraries for C (LibC)](http://developer.novell.com/ndk/apache.htm)
- [LDAP Libraries for C](http://developer.novell.com/ndk/apache.htm)
- [ZLIB Compression Library source code](http://developer.novell.com/ndk/apache.htm)
- AWK utility (awk, gawk or similar). AWK can be downloaded from [http://developer.novell.com/ndk/apache.htm](http://developer.novell.com/ndk/apache.htm). The utility must be found in your windows path and must be named awk.exe.

**Building Apache using the NetWare makefiles:**
- Set the environment variable NOVELLLIBC to the location of the NetWare Libraries for C SDK, for example:

  ```bash
  Set NOVELLLIBC=c:\novell\ndk\libc
  ```

- Set the environment variable METROWERKS to the location where you installed the Metrowerks CodeWarrior compiler, for example:

  ```bash
  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:

  ```bash
  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:

  ```bash
  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:

  ```bash
  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](https://www.openssl.org) (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
```
Running a High-Performance Web Server on HP-UX

Date: Wed, 05 Nov 1997 16:59:34 -0800
From: Rick Jones <raj@cup.hp.com>
Reply-To: raj@cup.hp.com
Organization: Network Performance
Subject: HP-UX tuning tips

Here are some tuning tips for HP-UX to add to the tuning page.

For HP-UX 9.X: Upgrade to 10.20
For HP-UX 10.[00|01|10]: Upgrade to 10.20

For HP-UX 10.20:

Install the latest cumulative ARPA Transport Patch. This will allow you to configure the size of the TCP connection lookup hash table. The default is 256 buckets and must be set to a power of two. This is accomplished with adb against the *disc* image of the kernel. The variable name is tcp_hash_size. Notice that it's critically important that you use "W" to write a 32 bit quantity, not "w" to write a 16 bit value when patching the disc image because the tcp_hash_size variable is a 32 bit quantity.

How to pick the value? Examine the output of ftp://ftp.cup.hp.com/dist/networking/tools/connhist and see how many total TCP connections exist on the system. You probably want that number divided by the hash table size to be reasonably small, say less than 10. Folks can look at HP's SPECweb96 disclosures for some common settings. These can be found at http://www.specbench.org/. If an HP-UX system was performing at 1000 SPECweb96 connections per second, the TIME_WAIT time of 60 seconds would mean 60,000 TCP "connections" being tracked.

If folks are running Apache on a PA-8000 based system, they should consider "chatr'ing" the Apache executable to have a large page size. This would be "chatr +pi L <BINARY>". The GID of the running executable must have MLOCK privileges. Setprivgrp(1m) should be consulted for assigning MLOCK. The change can be validated by running Glance and examining the memory regions of the server(s) to make sure that they show a non-trivial fraction of the text segment being locked.

If folks are running Apache on MP systems, they might consider writing a small program that uses mpctl() to bind processes to processors. A simple pid % numcpu algorithm is probably sufficient. This might even go into the source code.

If folks are concerned about the number of FIN_WAIT_2 connections, they can use nettune to shrink the value of tcp_keepstart. However, they should be careful there - certainly do not make it less than oh two to four minutes. If tcp_hash_size has been set well, it is probably OK to let the FIN_WAIT_2's take longer to timeout (perhaps even the default two hours) - they will not on average have a big impact on performance.

There are other things that could go into the code base, but that might be left for another email. Feel free to drop me a message if you or others are interested.

sincerely,

rick jones

http://www.netperf.org/netperf/
The Apache EBCDIC Port

**Warning:** This document has not been updated to take into account changes made in the 2.0 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.
Version 1.3 of the Apache HTTP Server was the first version which included a port to a (non-ASCII) mainframe machine which uses the EBCDIC character set as its native codeset.

(It is the SIEMENS family of mainframes running the BS2000/OSD operating system. This mainframe OS nowadays features a SVR4-derived POSIX subsystem).

The port was started initially to

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

This document serves as a rationale to describe some of the design decisions of the port to this machine.
One objective of the EBCDIC port was to maintain enough backwards compatibility with the (EBCDIC) CERN server to make the transition to the new server attractive and easy. This required the addition of a configurable method to define whether a HTML document was stored in ASCII (the only format accepted by the old server) or in EBCDIC (the native document format in the POSIX subsystem, and therefore the only realistic format in which the other POSIX tools like grep or sed could operate on the documents). The current solution to this is a "pseudo-MIME-format" which is intercepted and interpreted by the Apache server (see below). Future versions might solve the problem by defining an "ebcdic-handler" for all documents which must be converted.
Since all Apache input and output is based upon the BUFF data type and its methods, the easiest solution was to add the conversion to the BUFF handling routines. The conversion must be settable at any time, so a BUFF flag was added which defines whether a BUFF object has currently enabled conversion or not. This flag is modified at several points in the HTTP protocol:

- **set** before a request is received (because the request and the request header lines are always in ASCII format)
- **set/unset** when the request body is received - depending on the content type of the request body (because the request body may contain ASCII text or a binary file)
- **set** before a reply header is sent (because the response header lines are always in ASCII format)
- **set/unset** when the response body is sent - depending on the content type of the response body (because the response body may contain text or a binary file)
1. The relevant changes in the source are `#ifdef`'ed into two categories:

   **#ifdef CHARSET_EBCDIC**
   
   Code which is needed for any EBCDIC based machine. This includes character translations, differences in contiguity of the two character sets, flags which indicate which part of the HTTP protocol has to be converted and which part doesn't *etc*.

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

2. The possibility to translate between ASCII and EBCDIC at the socket level (on BS2000 POSIX, there is a socket option which supports this) was intentionally *not* chosen, because the byte stream at the HTTP protocol level consists of a mixture of protocol related strings and non-protocol related raw file data. HTTP protocol strings are always encoded in ASCII (the GET request, any Header: lines, the chunking information *etc.*) whereas the file transfer parts (*i.e.*, GIF images, CGI output *etc.*) should usually be just "passed through" by the server. This separation between "protocol string" and "raw data" is reflected in the server code by functions like `bgets()` or `rvputs()` for strings, and functions like `bwrite()` for binary data. A global translation of everything would therefore be inadequate.

   (In the case of text files of course, provisions must be made so that EBCDIC documents are always served in ASCII)
3. This port therefore features a built-in protocol level conversion for the server-internal strings (which the compiler translated to EBCDIC strings) and thus for all server-generated documents. The hard coded ASCII escapes \012 and \015 which are ubiquitous in the server code are an exception: they are already the binary encoding of the ASCII \n and \r and must not be converted to ASCII a second time. This exception is only relevant for server-generated strings; and external EBCDIC documents are not expected to contain ASCII newline characters.

4. By examining the call hierarchy for the BUFF management routines, I added an "ebcdic/ascii conversion layer" which would be crossed on every puts/write/get/gets, and a conversion flag which allowed enabling/disabling the conversions on-the-fly. Usually, a document crosses this layer twice from its origin source (a file or CGI output) to its destination (the requesting client): file -> Apache, and Apache -> client.

The server can now read the header lines of a CGI-script output in EBCDIC format, and then find out that the remainder of the script's output is in ASCII (like in the case of the output of a WWW Counter program: the document body contains a GIF image). All header processing is done in the native EBCDIC format; the server then determines, based on the type of document being served, whether the document body (except for the chunking information, of course) is in ASCII already or must be converted from EBCDIC.

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

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

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

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

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

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

8. For CGI output, the CGI script determines whether a conversion is needed or not: by setting the appropriate Content-Type, text files can be converted, or GIF output can be passed through unmodified. An example for the latter case is the wwwcount program which we ported as well.
Binary Files

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

When exchanging binary files between the mainframe host and a Unix machine or Windows PC, be sure to use the ftp "binary" (TYPE I) command, or use the rcp -b command from the mainframe host (the -b switch is not supported in unix rcp's).

Text Documents

The default assumption of the server is that Text Files (i.e., all files whose Content-Type: starts with text/) are stored in the native character set of the host, EBCDIC.

Server Side Included Documents

SSI documents must currently be stored in EBCDIC only. No provision is made to convert it from ASCII before processing.
<table>
<thead>
<tr>
<th>Module</th>
<th>Status</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>core</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_access</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_actions</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_alias</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_asis</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_auth</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_authn_anon</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_authn_dbm</td>
<td>?</td>
<td>with own libdb.a</td>
</tr>
<tr>
<td>mod_authz_dbm</td>
<td>?</td>
<td>with own libdb.a</td>
</tr>
<tr>
<td>mod_autoindex</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_cern_meta</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>mod_cgi</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_digest</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_dir</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_so</td>
<td>-</td>
<td>no shared libs</td>
</tr>
<tr>
<td>mod_env</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_example</td>
<td>-</td>
<td>(test bed only)</td>
</tr>
<tr>
<td>mod.expires</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_headers</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_imagemap</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_include</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_info</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_log_agent</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_log_config</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_log_referer</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_mime</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_mime_magic</td>
<td>?</td>
<td>not ported yet</td>
</tr>
<tr>
<td>Module</td>
<td>Status</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------</td>
<td>--------</td>
<td>---------</td>
</tr>
<tr>
<td>mod_negotiation</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_proxy</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_rewrite</td>
<td>+</td>
<td>untested</td>
</tr>
<tr>
<td>mod_setenvif</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_speling</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_status</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_unique_id</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_userdir</td>
<td>+</td>
<td></td>
</tr>
<tr>
<td>mod_usertrack</td>
<td>?</td>
<td>untested</td>
</tr>
<tr>
<td>Module</td>
<td>Status</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------</td>
<td>--------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td><strong>JK (Formerly mod_jserv)</strong></td>
<td>-</td>
<td>JAVA still being ported.</td>
</tr>
<tr>
<td><strong>mod_php3</strong></td>
<td>+</td>
<td>mod_php3 runs fine, with LDAP and GD and FreeType libraries.</td>
</tr>
<tr>
<td><strong>mod_put</strong></td>
<td>?</td>
<td>untested</td>
</tr>
<tr>
<td><strong>mod_session</strong></td>
<td>-</td>
<td>untested</td>
</tr>
</tbody>
</table>

Copyright 2017 The Apache Software Foundation. Licensed under the [Apache License, Version 2.0](https://www.apache.org/licenses/LICENSE-2.0).
httpd - Apache Hypertext Transfer Protocol Server

httpd is the Apache HyperText Transfer Protocol (HTTP) server program. It is designed to be run as a standalone daemon process. When used like this it will create a pool of child processes or threads to handle requests.

In general, httpd should not be invoked directly, but rather should be invoked via apachectl on Unix-based systems or as a service on Windows NT, 2000 and XP and as a console application on Windows 9x and ME.

See also

- Starting Apache httpd
- Stopping Apache httpd
- Configuration Files
- Platform-specific Documentation
- apachectl
httpd [ -d serverroot ] [ -f config ] [ -C directive ] [ -c directive ] [ -D parameter ] [ -e level ] [ -E file ] [ -k start|restart|graceful|stop|graceful-stop ] [ -h ] [ -l ] [ -L ] [ -S ] [ -t ] [ -v ] [ -V ] [ -X ] [ -M ] [ -T ]

On Windows systems, the following additional arguments are available:

httpd [ -k install|config|uninstall ] [ -n name ] [ -w ]
-d serverroot
   Set the initial value for the ServerRoot directive to serverroot. This can be overridden by the ServerRoot directive in the configuration file. The default is /usr/local/apache2.

-f config
   Uses the directives in the file config on startup. If config does not begin with a /, then it is taken to be a path relative to the ServerRoot. The default is conf/httpd.conf.

-k start|restart|graceful|stop|graceful-stop
   Signals httpd to start, restart, or stop. See Stopping Apache httpd for more information.

-C directive
   Process the configuration directive before reading config files.

-c directive
   Process the configuration directive after reading config files.

-D parameter
   Sets a configuration parameter which can be used with <IfDefine> sections in the configuration files to conditionally skip or process commands at server startup and restart. Also can be used to set certain less-common startup parameters including -DNO_DETACH (prevent the parent from forking) and -DFOREGROUND (prevent the parent from calling setsid() et al).

-e level
   Sets theLogLevel to level during server startup. This is useful for temporarily increasing the verbosity of the error messages to find problems during startup.

-E file
   Send error messages during server startup to file.
-h
Output a short summary of available command line options.

-1
Output a list of modules compiled into the server. This will not list dynamically loaded modules included using the LoadModule directive.

-L
Output a list of directives provided by static modules, together with expected arguments and places where the directive is valid. Directives provided by shared modules are not listed.

-M
Dump a list of loaded Static and Shared Modules.

-S
Show the settings as parsed from the config file (currently only shows the virtualhost settings).

-T (Available in 2.3.8 and later)
Skip document root check at startup/restart.

-t
Run syntax tests for configuration files only. The program immediately exits after these syntax parsing tests with either a return code of 0 (Syntax OK) or return code not equal to 0 (Syntax Error). If -D DUMP_VHOSTS is also set, details of the virtual host configuration will be printed. If -D DUMP_MODULES is set, all loaded modules will be printed.

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

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

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

The following arguments are available only on the Windows platform:

-k install|config|uninstall
Install Apache httpd as a Windows NT service; change startup options for the Apache httpd service; and uninstall the Apache httpd service.

-n name
The name of the Apache httpd service to signal.

-w
Keep the console window open on error so that the error message can be read.
ab - Apache HTTP server benchmarking tool

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

See also

httpd
ab [ -A auth-username:password ] [ -b windowsize ] [ -B local-address ] [ -c concurrency ] [ -C cookie-name=value ] [ -d ] [ -e csv-file ] [ -f protocol ] [ -g gnuplot-file ] [ -h ] [ -H custom-header ] [ -i ] [ -k ] [ -l ] [ -m HTTP-method ] [ -n requests ] [ -p POST-file ] [ -P proxy-auth-username:password ] [ -q ] [ -r ] [ -s timeout ] [ -S ] [ -t timelimint ] [ -T content-type ] [ -u PUT-file ] [ -v verbosity] [ -V ] [ -w ] [ -x <table>-attributes ] [ -X proxy[:port] ] [ -y <tr>-attributes ] [ -z <td>-attributes ] [ -Z ciphersuite ] [http[s]://]hostname[:port]/path
-A **auth-username:password**
  Supply BASIC Authentication credentials to the server. The username and password are separated by a single : and sent on the wire base64 encoded. The string is sent regardless of whether the server needs it (i.e., has sent an 401 authentication needed).

-b **windowsize**
Size of TCP send/receive buffer, in bytes.

-B **local-address**
Address to bind to when making outgoing connections.

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

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

-d
Do not display the "percentage served within XX [ms] table". (legacy support).

-e **csv-file**
Write a Comma separated value (CSV) file which contains for each percentage (from 1% to 100%) the time (in milliseconds) it took to serve that percentage of the requests. This is usually more useful than the 'gnuplot' file; as the results are already 'binned'.

-f **protocol**
Specify SSL/TLS protocol (SSL2, SSL3, TLS1, TLS1.1, TLS1.2, or ALL). TLS1.1 and TLS1.2 support available in 2.4.4 and later.

-g **gnuplot-file**
Write all measured values out as a 'gnuplot' or TSV (Tab separate values) file. This file can easily be imported into packages like Gnuplot, IDL, Mathematica, Igor or even Excel. The labels are on the first line of the file.

-h
Display usage information.

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

-i
Do HEAD requests instead of GET.

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

-l
Do not report errors if the length of the responses is not constant. This can be useful for dynamic pages. Available in 2.4.7 and later.

-m HTTP-method
Custom HTTP method for the requests. Available in 2.4.10 and later.

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

-p POST-file
File containing data to POST. Remember to also set -T.

-P proxy-auth-username:password
Supply BASIC Authentication credentials to a proxy en-route. The username and password are separated by a single : and sent on the wire base64 encoded. The string is sent regardless of whether the proxy needs it (i.e., has sent an 407 proxy authentication needed).

\textbf{\texttt{-q}}

When processing more than 150 requests, ab outputs a progress count on stderr every 10% or 100 requests or so. The -q flag will suppress these messages.

\textbf{\texttt{-r}}

Don't exit on socket receive errors.

\textbf{\texttt{-s \textit{timeout}}}

Maximum number of seconds to wait before the socket times out. Default is 30 seconds. Available in 2.4.4 and later.

\textbf{\texttt{-S}}

Do not display the median and standard deviation values, nor display the warning/error messages when the average and median are more than one or two times the standard deviation apart. And default to the min/avg/max values. (legacy support).

\textbf{\texttt{-t \textit{timelimit}}}

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

\textbf{\texttt{-T \textit{content-type}}}

Content-type header to use for POST/PUT data, eg. application/x-www-form-urlencoded. Default is text/plain.

\textbf{\texttt{-u PUT-file}}

File containing data to PUT. Remember to also set -T.
-v *verbosity*
  Set verbosity level - 4 and above prints information on headers, 3 and above prints response codes (404, 200, etc.), 2 and above prints warnings and info.

-V
  Display version number and exit.

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

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

-X *proxy[:port]*
  Use a proxy server for the requests.

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

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

-Z *ciphersuite*
  Specify SSL/TLS cipher suite (See openssl ciphers)
The following list describes the values returned by ab:

**Server Software**
The value, if any, returned in the server HTTP header of the first successful response. This includes all characters in the header from beginning to the point a character with decimal value of 32 (most notably: a space or CR/LF) is detected.

**Server Hostname**
The DNS or IP address given on the command line

**Server Port**
The port to which ab is connecting. If no port is given on the command line, this will default to 80 for http and 443 for https.

**SSL/TLS Protocol**
The protocol parameters negotiated between the client and server. This will only be printed if SSL is used.

**Document Path**
The request URI parsed from the command line string.

**Document Length**
This is the size in bytes of the first successfully returned document. If the document length changes during testing, the response is considered an error.

**Concurrency Level**
The number of concurrent clients used during the test

**Time taken for tests**
This is the time taken from the moment the first socket connection is created to the moment the last response is received

**Complete requests**
The number of successful responses received

**Failed requests**
The number of requests that were considered a failure. If the number is greater than zero, another line will be printed showing the number of requests that failed due to connecting, reading, incorrect content length, or exceptions.

**Write errors**
The number of errors that failed during write (broken pipe).

**Non-2xx responses**
The number of responses that were not in the 200 series of response codes. If all responses were 200, this field is not printed.

**Keep-Alive requests**
The number of connections that resulted in Keep-Alive requests

**Total body sent**
If configured to send data as part of the test, this is the total number of bytes sent during the tests. This field is omitted if the test did not include a body to send.

**Total transferred**
The total number of bytes received from the server. This number is essentially the number of bytes sent over the wire.

**HTML transferred**
The total number of document bytes received from the server. This number excludes bytes received in HTTP headers

**Requests per second**
This is the number of requests per second. This value is the result of dividing the number of requests by the total time taken

**Time per request**
The average time spent per request. The first value is calculated with the formula concurrency * timetaken * 1000 / done while the second value is calculated with the
Transfer rate
The rate of transfer as calculated by the formula
\[
\frac{\text{totalread}}{1024} / \text{timetaken}
\]
There are various statically declared buffers of fixed length. Combined with the lazy parsing of the command line arguments, the response headers from the server and other external inputs, this might bite you.

It does not implement HTTP/1.x fully; only accepts some 'expected' forms of responses. The rather heavy use of `strstr(3)` shows up top in profile, which might indicate a performance problem; i.e., you would measure the `ab` performance rather than the server's.
apachectl is a front end to the Apache HyperText Transfer Protocol (HTTP) server. It is designed to help the administrator control the functioning of the Apache httpd daemon.

The apachectl script can operate in two modes. First, it can act as a simple front-end to the httpd command that simply sets any necessary environment variables and then invokes httpd, passing through any command line arguments. Second, apachectl can act as a SysV init script, taking simple one-word arguments like start, restart, and stop, and translating them into appropriate signals to httpd.

If your Apache installation uses non-standard paths, you will need to edit the apachectl script to set the appropriate paths to the httpd binary. You can also specify any necessary httpd command line arguments. See the comments in the script for details.

The apachectl script returns a 0 exit value on success, and >0 if an error occurs. For more details, view the comments in the script.

See also

Starting Apache
Stopping Apache
Configuration Files
Platform Docs
httpd
When acting in pass-through mode, apachectl can take all the arguments available for the `httpd` binary.

`apachectl [ httpd-argument ]`

When acting in SysV init mode, apachectl takes simple, one-word commands, defined below.

`apachectl command`
Only the SysV init-style options are defined here. Other arguments are defined on the httpd manual page.

**start**
Start the Apache httpd daemon. Gives an error if it is already running. This is equivalent to apachectl -k start.

**stop**
Stops the Apache httpd daemon. This is equivalent to apachectl -k stop.

**restart**
Restarts the Apache httpd daemon. If the daemon is not running, it is started. This command automatically checks the configuration files as in configtest before initiating the restart to make sure the daemon doesn't die. This is equivalent to apachectl -k restart.

**fullstatus**
Displays a full status report from mod_status. For this to work, you need to have mod_status enabled on your server and a text-based browser such as lynx available on your system. The URL used to access the status report can be set by editing the STATUSURL variable in the script.

**status**
Displays a brief status report. Similar to the fullstatus option, except that the list of requests currently being served is omitted.

**graceful**
Gracefully restarts the Apache httpd daemon. If the daemon is not running, it is started. This differs from a normal restart in that currently open connections are not aborted. A side effect
is that old log files will not be closed immediately. This means that if used in a log rotation script, a substantial delay may be necessary to ensure that the old log files are closed before processing them. This command automatically checks the configuration files as in configtest before initiating the restart to make sure Apache doesn't die. This is equivalent to apachectl -k graceful.

**graceful-stop**
Gracefully stops the Apache httpd daemon. This differs from a normal stop in that currently open connections are not aborted. A side effect is that old log files will not be closed immediately. This is equivalent to apachectl -k graceful-stop.

**configtest**
Run a configuration file syntax test. It parses the configuration files and either reports Syntax 0k or detailed information about the particular syntax error. This is equivalent to apachectl -t.

The following option was available in earlier versions but has been removed.

**startssl**
To start httpd with SSL support, you should edit your configuration file to include the relevant directives and then use the normal apachectl start.
apxs - APache eXtenSion tool

apxs is a tool for building and installing extension modules for the Apache HyperText Transfer Protocol (HTTP) server. This is achieved by building a dynamic shared object (DSO) from one or more source or object files which then can be loaded into the Apache server under runtime via the `LoadModule` directive from `mod_so`.

So to use this extension mechanism your platform has to support the DSO feature and your Apache `httpd` binary has to be built with the `mod_so` module. The apxs tool automatically complains if this is not the case. You can check this yourself by manually running the command

```
$ httpd -l
```

The module `mod_so` should be part of the displayed list. If these requirements are fulfilled you can easily extend your Apache server's functionality by installing your own modules with the DSO mechanism by the help of this apxs tool:

```
$ apxs -i -a -c mod_foo.c
gcc -fpic -DSHARED_MODULE -I/path/to/apache/include -c mod_foo.c
ld -Bshareable -o mod_foo.so mod_foo.o
cp mod_foo.so /path/to/apache/modules/mod_foo.so
chmod 755 /path/to/apache/modules/mod_foo.so
[activating module 'foo' in /path/to/apache/etc/httpd.conf]
$ apachectl restart
/path/to/apache/sbin/apachectl restart: httpd not running, trying to start
/path/to/apache/sbin/apachectl restart: httpd started
$ _
```

The arguments `files` can be any C source file (.c), a object file (.o) or even a library archive (.a). The apxs tool automatically recognizes
these extensions and automatically used the C source files for compilation while just using the object and archive files for the linking phase. But when using such pre-compiled objects make sure they are compiled for position independent code (PIC) to be able to use them for a dynamically loaded shared object. For instance with GCC you always just have to use -fpic. For other C compilers consult its manual page or at watch for the flags apxs uses to compile the object files.

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

See also

apachectl
httpd
apxs -g [ -S name=value ] -n modname

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

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

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

apxs -e [ -S name=value ] [ -n modname ] [ -a ] [ -A ] dso-file ...
Common Options

-\(n\) modname

This explicitly sets the module name for the -\(i\) (install) and -\(g\) (template generation) option. Use this to explicitly specify the module name. For option -\(g\) this is required, for option -\(i\) the apxs tool tries to determine the name from the source or (as a fallback) at least by guessing it from the filename.

Query Options

-\(q\)

Performs a query for variables and environment settings used to build httpd. When invoked without query parameters, it prints all known variables and their values. The optional -\(v\) parameter formats the list output.

Use this to manually determine settings used to build the httpd that will load your module. For instance use

```
INC=-I`apxs -q INCLUDEDIR`
```

inside your own Makefiles if you need manual access to Apache's C header files.

Configuration Options

-\(S\) name=value

This option changes the apxs settings described above.

Template Generation Options

-\(g\)

This generates a subdirectory name (see option -\(n\)) and there two files: A sample module source file named mod_name.c
which can be used as a template for creating your own modules or as a quick start for playing with the apxs mechanism. And a corresponding Makefile for even easier build and installing of this module.

**DSO Compilation Options**

**-c**
This indicates the compilation operation. It first compiles the C source files (.c) of `files` into corresponding object files (.o) and then builds a dynamically shared object in `dsofile` by linking these object files plus the remaining object files (.o and .a) of `files`. If no -o option is specified the output file is guessed from the first filename in `files` and thus usually defaults to `mod_name.so`.

**-o dsofile**
Explicitly specifies the filename of the created dynamically shared object. If not specified and the name cannot be guessed from the `files` list, the fallback name `mod_unknown.so` is used.

**-D name=value**
This option is directly passed through to the compilation command(s). Use this to add your own defines to the build process.

**-I incdir**
This option is directly passed through to the compilation command(s). Use this to add your own include directories to search to the build process.

**-L libdir**
This option is directly passed through to the linker command. Use this to add your own library directories to search to the build process.
-l **libname**
   This option is directly passed through to the linker command. Use this to add your own libraries to search to the build process.

-Wc, **compiler-flags**
   This option passes `compiler-flags` as additional flags to the `libtool --mode=compile` command. Use this to add local compiler-specific options.

-Wl, **linker-flags**
   This option passes `linker-flags` as additional flags to the `libtool --mode=link` command. Use this to add local linker-specific options.

-p
   This option causes `apxs` to link against the apr/apr-util libraries. This is useful when compiling helper programs that use the apr/apr-util libraries.

**DSO Installation and Configuration Options**

-i
   This indicates the installation operation and installs one or more dynamically shared objects into the server's `modules` directory.

-a
   This activates the module by automatically adding a corresponding `LoadModule` line to Apache's `httpd.conf` configuration file, or by enabling it if it already exists.

-A
   Same as option -a but the created `LoadModule` directive is prefixed with a hash sign (#), *i.e.*, the module is just prepared for later activation but initially disabled.

-e
This indicates the editing operation, which can be used with the -a and -A options similarly to the -i operation to edit Apache's httpd.conf configuration file without attempting to install the module.
Assume you have an Apache module named `mod_foo.c` available which should extend Apache’s server functionality. To accomplish this you first have to compile the C source into a shared object suitable for loading into the Apache server under runtime via the following command:

```
$ apxs -c mod_foo.c
/path/to/libtool --mode=compile gcc ... -c mod_foo.c
/path/to/libtool --mode=link gcc ... -o mod_foo.la mod_foo.slo
```

Then you have to update the Apache configuration by making sure a `LoadModule` directive is present to load this shared object. To simplify this step `apxs` provides an automatic way to install the shared object in its "modules" directory and updating the `httpd.conf` file accordingly. This can be achieved by running:

```
$ apxs -i -a mod_foo.la
/path/to/instdso.sh mod_foo.la /path/to/apache/modules
/path/to/libtool --mode=install cp mod_foo.la
/path/to/apache/modules ... chmod 755
/path/to/apache/modules/mod_foo.so
[activating module 'foo' in /path/to/apache/conf/httpd.conf]
```

This way a line named

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

is added to the configuration file if still not present. If you want to have this disabled per default use the `-A` option, i.e.

```
$ apxs -i -A mod_foo.c
```

For a quick test of the `apxs` mechanism you can create a sample Apache module template plus a corresponding Makefile via:
Then you can immediately compile this sample module into a shared object and load it into the Apache server:

$ cd foo
$ make all reload
   apxs -c mod_foo.c
   /path/to/libtool --mode=compile gcc ... -c mod_foo.c
   /path/to/libtool --mode=link gcc ... -o mod_foo.la mod_foo.slo
   apxs -i -a -n "foo" mod_foo.la
   /path/to/instdso.sh mod_foo.la /path/to/apache/modules
   /path/to/libtool --mode=install cp mod_foo.la
   /path/to/apache/modules ... chmod 755
   /path/to/apache/modules/mod_foo.so
   [activating module `foo' in /path/to/apache/conf/httpd.conf]
   apachectl restart
   /path/to/apache/sbin/apachectl restart: httpd not running, trying to start
   /path/to/apache/sbin/apachectl restart: httpd started
$ _
configure - Configure the source tree

The configure script configures the source tree for compiling and installing the Apache HTTP Server on your particular platform. Various options allow the compilation of a server corresponding to your personal requirements.

This script, included in the root directory of the source distribution, is for compilation on Unix and Unix-like systems only. For other platforms, see the [platform documentation](#).

See also

[Compiling and Installing](#)
You should call the configure script from within the root directory of the distribution.

`.configure [OPTION]... [VAR=VALUE]...`

To assign environment variables (e.g. CC, CFLAGS ...), specify them as `VAR=VALUE`. See below for descriptions of some of the useful variables.
Configuration options

The following options influence the behavior of configure itself.

-C

--config-cache
    This is an alias for --cache-file=config.cache

--cache-file=FILE
    The test results will be cached in file FILE. This option is disabled by default.

-h

--help [short|recursive]
    Output the help and exit. With the argument short only options specific to this package will displayed. The argument recursive displays the short help of all the included packages.

-n

--no-create
    The configure script is run normally but does not create output files. This is useful to check the test results before generating makefiles for compilation.

-q

--quiet
    Do not print checking ... messages during the configure
process.

--srcdir=DIR
Defines directory DIR to be the source file directory. Default is the directory where configure is located, or the parent directory.

--silent
Same as --quiet

-V

--version
Display copyright information and exit.

Installation directories
These options define the installation directory. The installation tree depends on the selected layout.

--prefix=PREFIX
Install architecture-independent files in PREFIX. By default the installation directory is set to /usr/local/apache2.

--exec-prefix=EPREFIX
Install architecture-dependent files in EPREFIX. By default the installation directory is set to the PREFIX directory.

By default, make install will install all the files in /usr/local/apache2/bin, /usr/local/apache2/lib etc. You can specify an installation prefix other than /usr/local/apache2 using --prefix, for instance --prefix=$HOME.

Define a directory layout
--enable-layout=LAYOUT
Configure the source code and build scripts to assume an
installation tree based on the layout \textit{LAYOUT}. This allows you to separately specify the locations for each type of file within the Apache HTTP Server installation. The \texttt{config.layout} file contains several example configurations, and you can also create your own custom configuration following the examples. The different layouts in this file are grouped into \texttt{<Layout F00>...</Layout>} sections and referred to by name as in F00. The default layout is Apache.

\textbf{Fine tuning of the installation directories}

For better control of the installation directories, use the options below. Please note that the directory defaults are set by autoconf and are overwritten by the corresponding layout setting.

\texttt{--bindir=DIR}

Install user executables in\texttt{DIR}. The user executables are supporting programs like \texttt{htpasswd}, \texttt{dbmmanage}, etc. which are useful for site administrators. By default \texttt{DIR} is set to \texttt{EPREFIX/bin}.

\texttt{--datadir=DIR}

Install read-only architecture-independent data in \texttt{DIR}. By default datadir is set to \texttt{PREFIX/share}. This option is offered by autoconf and currently unused.

\texttt{--includedir=DIR}

Install C header files in \texttt{DIR}. By default includedir is set to \texttt{EPREFIX/include}.

\texttt{--infodir=DIR}

Install info documentation in \texttt{DIR}. By default infodir is set to \texttt{PREFIX/info}. This option is currently unused.

\texttt{--libdir=DIR}

Install object code libraries in \texttt{DIR}. By default libdir is set to
Install the program executables (i.e., shared modules) in $DIR$. By default libexecdir is set to $EPREFIX/lib$.

```
--libexecdir=$DIR
```

Install modifiable single-machine data in $DIR$. By default localstatedir is set to $PREFIX/var$. This option is offered by autoconf and currently unused.

```
--localstatedir=$DIR
```

Install the man documentation in $DIR$. By default mandir is set to $EPREFIX/man$.

```
--mandir=$DIR
```

Install C header files for non-gcc in $DIR$. By default oldincludedir is set to /usr/include. This option is offered by autoconf and currently unused.

```
--oldincludedir=$DIR
```

Install the system administrator executables in $DIR$. Those are server programs like httpd, apachectl, suexec, etc. which are necessary to run the Apache HTTP Server. By default sbindir is set to $EPREFIX/sbin$.

```
--sbindir=$DIR
```

Install modifiable architecture-independent data in $DIR$. By default sharedstatedir is set to $PREFIX/com$. This option is offered by autoconf and currently unused.

```
--sharedstatedir=$DIR
```

Install read-only single-machine data like the server configuration files httpd.conf, mime.types, etc. in $DIR$. By default sysconfdir is set to $PREFIX/conf$.

```
--sysconfdir=$DIR
```

System types
These options are used to cross-compile the Apache HTTP Server to run on another system. In normal cases, when building and running the server on the same system, these options are not used.

--build=BUILD
Defines the system type of the system on which the tools are being built. It defaults to the result of the script `config.guess`.

--host=HOST
Defines the system type of the system on which the server will run. HOST defaults to BUILD.

--target=TARGET
Configure for building compilers for the system type TARGET. It defaults to HOST. This option is offered by autoconf and not necessary for the Apache HTTP Server.

**Optional Features**

These options are used to fine tune the features your HTTP server will have.

**General syntax**

Generally you can use the following syntax to enable or disable a feature:

--disable-FEATURE
Do not include FEATURE. This is the same as --enable-FEATURE=no.

--enable-FEATURE[=ARG]
Include FEATURE. The default value for ARG is yes.

--enable-MODULE=shared
The corresponding module will be build as DSO module. By
default enabled modules are linked dynamically.

**--enable-\textsc{MODULE}=static**

The corresponding module will be linked statically.

**Note**

`configure` will not complain about `--enable-foo` even if `foo` doesn't exist, so you need to type carefully.

**Choosing modules to compile**

Most modules are compiled by default and have to be disabled explicitly or by using the keyword `few` (see `--enable-modules`, `--enable-mods-shared` and `--enable-mods-static` below for further explanation) or `--enable-modules=none` to be removed as a group.

Other modules are not compiled by default and have to be enabled explicitly or by using the keywords `all` or `reallyall` to be available.

To find out which modules are compiled by default, run `./configure -h` or `./configure --help` and look under `Optional Features`. Suppose you are interested in `mod_example1` and `mod_example2`, and you see this:

```
Optional Features:
...
--disable-example1    example module 1
--enable-example2    example module 2
...
```

Then `mod_example1` is enabled by default, and you would use `--disable-example1` to not compile it. `mod_example2` is disabled by default, and you would use `--enable-example2` to compile it.
**Multi-Processing Modules**

Multi-Processing Modules, or MPMs, implement the basic behavior of the server. A single MPM must be active in order for the server to function. The list of available MPMs appears on the module index page.

MPMs can be built as DSOs for dynamic loading or statically linked with the server, and are enabled using the following options:

---

**--with-mpm=MPM**

Choose the default MPM for your server. If MPMs are built as DSO modules (see --enable-mpms-shared), this directive selects the MPM which will be loaded in the default configuration file. Otherwise, this directive selects the only available MPM, which will be statically linked into the server.

If this option is omitted, the default MPM for your operating system will be used.

**--enable-mpms-shared=MPM-LIST**

Enable a list of MPMs as dynamic shared modules. One of these modules must be loaded dynamically using the LoadModule directive.

`MPM-LIST` is a space-separated list of MPM names enclosed by quotation marks. For example:

```
--enable-mpms-shared='prefork worker'
```

Additionally you can use the special keyword `all`, which will select all MPMs which support dynamic loading on the current platform and build them as DSO modules. For example:

```
--enable-mpms-shared=all
```
Third-party modules
To add additional third-party modules use the following options:

--with-module=module-type:module-file[, module-type:module-file]
Add one or more third-party modules to the list of statically linked modules. The module source file *module-file* will be searched in the modules/*module-type* subdirectory of your Apache HTTP server source tree. If it is not found there configure is considering *module-file* to be an absolute file path and tries to copy the source file into the *module-type* subdirectory. If the subdirectory doesn't exist it will be created and populated with a standard Makefile.in.

This option is useful to add small external modules consisting of one source file. For more complex modules you should read the vendor's documentation.

Note
If you want to build a DSO module instead of a statically linked use `apxs`.

Cumulative and other options
--enable-maintainer-mode
Turn on debugging and compile time warnings and load all compiled modules.

--enable-mods-shared=MODULE-LIST
Defines a list of modules to be enabled and build as dynamic shared modules. This mean, these module have to be loaded dynamically by using the LoadModule directive.

*MODULE-LIST* is a space separated list of modulenames enclosed by quotation marks. The module names are given
without the preceding mod_. For example:

```bash
--enable-mods-shared='headers rewrite dav'
```

Additionally you can use the special keywords reallyall, all, most and few. For example,

```bash
--enable-mods-shared=most
```

will compile most modules and build them as DSO modules,

```bash
--enable-mods-shared=few
```

will only compile a very basic set of modules.

The default set is most.

The `LoadModule` directives for the chosen modules will be automatically generated in the main configuration file. By default, all those directives will be commented out except for the modules that are either required or explicitly selected by a configure `--enable-foo` argument. You can change the set of loaded modules by activating or deactivating the `LoadModule` directives in `httpd.conf`. In addition the `LoadModule` directives for all built modules can be activated via the configure option `--enable-load-all-modules`.

```bash
--enable-mods-static=MODULE-LIST
```

This option behaves similar to `--enable-mods-shared`, but will link the given modules statically. This mean, these modules will always be present while running `httpd`. They need not be loaded with `LoadModule`.

```bash
--enable-modules=MODULE-LIST
```
This option behaves like to 
\texttt{--enable-mods-shared}, and will also link the given modules dynamically. The special keyword \texttt{none} disables the build of all modules.

\texttt{--enable-v4-mapped}
Allow IPv6 sockets to handle IPv4 connections.

\texttt{--with-port=PORT}
This defines the port on which \texttt{httpd} will listen. This port number is used when generating the configuration file \texttt{httpd.conf}. The default is 80.

\texttt{--with-program-name}
Define an alternative executable name. The default is \texttt{httpd}.

Optional packages
These options are used to define optional packages.

General syntax
Generally you can use the following syntax to define an optional package:

\texttt{--with-PACKAGE[=ARG]}
Use the package \texttt{PACKAGE}. The default value for \texttt{ARG} is \texttt{yes}.

\texttt{--without-PACKAGE}
Do not use the package \texttt{PACKAGE}. This is the same as \texttt{--with-PACKAGE=no}. This option is provided by \texttt{autoconf} but not very useful for the Apache HTTP Server.

Specific packages
\texttt{--with-apr=DIR|FILE}
The \texttt{Apache Portable Runtime} (APR) is part of the \texttt{httpd} source distribution and will automatically be build together
with the HTTP server. If you want to use an already installed APR instead you have to tell configure the path to the apr-config script. You may set the absolute path and name or the directory to the installed APR. apr-config must exist within this directory or the subdirectory bin.

--with-apr-util=DIR|FILE
The Apache Portable Runtime Utilities (APU) are part of the httpd source distribution and will automatically be build together with the HTTP server. If you want to use an already installed APU instead you have to tell configure the path to the apu-config script. You may set the absolute path and name or the directory to the installed APU. apu-config must exist within this directory or the subdirectory bin.

--with-ssl=DIR
If mod_ssl has been enabled configure searches for an installed OpenSSL. You can set the directory path to the SSL/TLS toolkit instead.

--with-z=DIR
configure searches automatically for an installed zlib library if your source configuration requires one (e.g., when mod_deflate is enabled). You can set the directory path to the compression library instead.

Several features of the Apache HTTP Server, including mod_authn_dbm and mod_rewrite's DBM RewriteMap use simple key/value databases for quick lookups of information. SDBM is included in the APU, so this database is always available. If you would like to use other database types, use the following options to enable them:

--with-gdbm[=path]
If no path is specified, configure will search for the include
files and libraries of a GNU DBM installation in the usual search paths. An explicit path will cause configure to look in path/lib and path/include for the relevant files. Finally, the path may specify specific include and library paths separated by a colon.

--with-ndbm[=path]
Like --with-gdbm, but searches for a New DBM installation.

--with-berkeley-db[=path]
Like --with-gdbm, but searches for a Berkeley DB installation.

Note
The DBM options are provided by the APU and passed through to its configuration script. They are useless when using an already installed APU defined by --with-apr-util.

You may use more than one DBM implementation together with your HTTP server. The appropriate DBM type will be configured within the runtime configuration at each time.

Options for support programs
--enable-static-support
Build a statically linked version of the support binaries. This means, a stand-alone executable will be built with all the necessary libraries integrated. Otherwise the support binaries are linked dynamically by default.

--enable-suexec
Use this option to enable suexec, which allows you to set uid and gid for spawned processes. Do not use this option unless you understand all the security implications of running a suid binary on your server. Further options to configure suexec are described below.
It is possible to create a statically linked binary of a single support program by using the following options:

--enable-static-ab
    Build a statically linked version of ab.

--enable-static-checkgid
    Build a statically linked version of checkgid.

--enable-static-htdbm
    Build a statically linked version of htdbm.

--enable-static-htdigest
    Build a statically linked version of htdigest.

--enable-static-htpasswd
    Build a statically linked version of htpasswd.

--enable-static-logresolve
    Build a statically linked version of logresolve.

--enable-static-rotatelogs
    Build a statically linked version of rotatelogs.

suexec configuration options
The following options are used to fine tune the behavior of suexec. See Configuring and installing suEXEC for further information.

--with-suexec-bin
    This defines the path to suexec binary. Default is --sbindir (see Fine tuning of installation directories).

--with-suexec-caller
    This defines the user allowed to call suexec. It should be the same as the user under which httpd normally runs.

--with-suexec-docroot
This defines the directory tree under which suexec access is allowed for executables. Default value is --datadir/htdocs.

--with-suexec-gidmin
Define this as the lowest GID allowed to be a target user for suexec. The default value is 100.

--with-suexec-logfile
This defines the filename of the suexec logfile. By default the logfile is named suexec_log and located in --logfiledir.

--with-suexec-safepath
Define the value of the environment variable PATH to be set for processes started by suexec. Default value is /usr/local/bin:/usr/bin:/bin.

--with-suexec-userdir
This defines the subdirectory under the user's directory that contains all executables for which suexec access is allowed. This setting is necessary when you want to use suexec together with user-specific directories (as provided by mod_userdir). The default is public_html.

--with-suexec-uidmin
Define this as the lowest UID allowed to be a target user for suexec. The default value is 100.

--with-suexec-umask
Set umask for processes started by suexec. It defaults to your system settings.
There are some useful environment variables to override the choices made by configure or to help it to find libraries and programs with nonstandard names or locations.

**CC**
Define the C compiler command to be used for compilation.

**CFLAGS**
Set C compiler flags you want to use for compilation.

**CPP**
Define the C preprocessor command to be used.

**CPPFLAGS**
Set C/C++ preprocessor flags, e.g. `-Iincludedir` if you have headers in a nonstandard directory `includedir`.

**LDFLAGS**
Set linker flags, e.g. `-Llibdir` if you have libraries in a nonstandard directory `libdir`. 
dbmmanage - Manage user authentication files in DBM format

dbmmanage is used to create and update the DBM format files used to store usernames and password for basic authentication of HTTP users via mod_authn_dbm. Resources available from the Apache HTTP server can be restricted to just the users listed in the files created by dbmmanage. This program can only be used when the usernames are stored in a DBM file. To use a flat-file database see htpasswd.

Another tool to maintain a DBM password database is htdbm.

This manual page only lists the command line arguments. For details of the directives necessary to configure user authentication in httpd see the httpd manual, which is part of the Apache distribution or can be found at http://httpd.apache.org/.

See also
  httpd
  htdbm
  mod_authn_dbm
  mod_authz_dbm
dbmmanage [ encoding ] filename
add|adduser|check|delete|update username [
encpasswd [ group[,group...] [ comment ] ] ]

dbmmanage filename view [ username ]

dbmmanage filename import
**filename**
The filename of the DBM format file. Usually without the extension `.db`, `.pag`, or `.dir`.

**username**
The user for which the operations are performed. The `username` may not contain a colon (`:`).

**encpasswd**
This is the already encrypted password to use for the `update` and `add` commands. You may use a hyphen (`-`) if you want to get prompted for the password, but fill in the fields afterwards. Additionally when using the update command, a period (`.`) keeps the original password untouched.

**group**
A group, which the user is member of. A groupname may not contain a colon (`:`). You may use a hyphen (`-`) if you don't want to assign the user to a group, but fill in the comment field. Additionally when using the update command, a period (`.`) keeps the original groups untouched.

**comment**
This is the place for your opaque comments about the user, like `realname`, `mailaddress` or such things. The server will ignore this field.

**Encodings**

- **-d**
crypt encryption (default, except on Win32, Netware)

- **-m**
MD5 encryption (default on Win32, Netware)

- **-s**
SHA1 encryption

-p
plaintext (not recommended)

Commands

add
Adds an entry for username to filename using the encrypted password encpasswd.

```
dbmmanage passwords.dat add rbowen foKntnEF3Ksxa
```

adduser
Asks for a password and then adds an entry for username to filename.

```
dbmmanage passwords.dat adduser krietz
```

check
Asks for a password and then checks if username is in filename and if it's password matches the specified one.

```
dbmmanage passwords.dat check rbowen
```

delete
Deletes the username entry from filename.

```
dbmmanage passwords.dat delete rbowen
```

import
Reads username:password entries (one per line) from stdin and adds them to filename. The passwords already have to be crypted.

update
Same as the adduser command, except that it makes sure *username* already exists in *filename*.

```
dbmmanage passwords.dat update rbowen
```

**view**

Just displays the contents of the DBM file. If you specify a *username*, it displays the particular record only.

```
dbmmanage passwords.dat view
```
One should be aware that there are a number of different DBM file formats in existence, and with all likelihood, libraries for more than one format may exist on your system. The three primary examples are SDBM, NDBM, the GNU project's GDBM, and Berkeley DB 2. Unfortunately, all these libraries use different file formats, and you must make sure that the file format used by filename is the same format that dbmmanage expects to see. dbmmanage currently has no way of determining what type of DBM file it is looking at. If used against the wrong format, will simply return nothing, or may create a different DBM file with a different name, or at worst, it may corrupt the DBM file if you were attempting to write to it.

dbmmanage has a list of DBM format preferences, defined by the @AnyDBM::ISA array near the beginning of the program. Since we prefer the Berkeley DB 2 file format, the order in which dbmmanage will look for system libraries is Berkeley DB 2, then NDBM, then GDBM and then SDBM. The first library found will be the library dbmmanage will attempt to use for all DBM file transactions. This ordering is slightly different than the standard @AnyDBM::ISA ordering in Perl, as well as the ordering used by the simple dbmopen( ) call in Perl, so if you use any other utilities to manage your DBM files, they must also follow this preference ordering. Similar care must be taken if using programs in other languages, like C, to access these files.

One can usually use the file program supplied with most Unix systems to see what format a DBM file is in.
htcacheclean - Clean up the disk cache

htcacheclean is used to keep the size of mod_cache_disk's storage within a given size limit, or limit on inodes in use. This tool can run either manually or in daemon mode. When running in daemon mode, it sleeps in the background and checks the cache directory at regular intervals for cached content to be removed. You can stop the daemon cleanly by sending it a TERM or INT signal. When run manually, a once off check of the cache directory is made for cached content to be removed. If one or more URLs are specified, each URL will be deleted from the cache, if present.

See also

mod_cache_disk
htcacheclean [ -D ] [ -v ] [ -t ] [ -r ] [ -n ] [ -Rround ] -ppath [-llimit| -Llimit]

htcacheclean [ -n ] [ -t ] [ -i ] [ -Ppidfile ] [ -Rround ] -dinterval -ppath [-llimit| -Llimit]

htcacheclean [ -v ] [ -Rround ] -ppath [ -a ] [ -A ]

htcacheclean [ -D ] [ -v ] [ -t ] [ -Rround ] -ppath url
**-d**

Daemonize and repeat cache cleaning every *interval* minutes. This option is mutually exclusive with the -D, -v and -r options. To shutdown the daemon cleanly, just send it a SIGTERM or SIGINT.

**-D**

Do a dry run and don't delete anything. This option is mutually exclusive with the -d option. When doing a dry run and deleting directories with -t, the inodes reported deleted in the stats cannot take into account the directories deleted, and will be marked as an estimate.

**-v**

Be verbose and print statistics. This option is mutually exclusive with the -d option.

**-r**

Clean thoroughly. This assumes that the Apache web server is not running (otherwise you may get garbage in the cache). This option is mutually exclusive with the -d option and implies the -t option.

**-n**

Be nice. This causes slower processing in favour of other processes. htcacheclean will sleep from time to time so that (a) the disk IO will be delayed and (b) the kernel can schedule other processes in the meantime.

**-t**

Delete all empty directories. By default only cache files are removed, however with some configurations the large number of directories created may require attention. If your configuration requires a very large number of directories, to the point that inode or file allocation table exhaustion may
become an issue, use of this option is advised.

**-p**
Specify *path* as the root directory of the disk cache. This should be the same value as specified with the `CacheRoot` directive.

**-P**
Specify *pidfile* as the name of the file to write the process ID to when daemonized.

**-R**
Specify *round* as the amount to round sizes up to, to compensate for disk block sizes. Set to the block size of the cache partition.

**-L**
Specify *limit* as the total disk cache size limit. The value is expressed in bytes by default (or attaching B to the number). Attach K for Kbytes or M for MBytes.

**-L**
Specify *limit* as the total disk cache inode limit.

**-i**
Be intelligent and run only when there was a modification of the disk cache. This option is only possible together with the -d option.

**-a**
List the URLs currently stored in the cache. Variants of the same URL will be listed once for each variant.

**-A**
List the URLs currently stored in the cache, along with their attributes in the following order: url, header size, body size, status, entity version, date, expiry, request time, response time, body present, head request.
If `htcacheclean` is passed one or more URLs, each URL will be deleted from the cache. If multiple variants of an URL exists, all variants would be deleted.

When a reverse proxied URL is to be deleted, the effective URL is constructed from the `Host` header, the `port`, the `path` and the `query`. Note the '?' in the URL must always be specified explicitly, whether a query string is present or not. For example, an attempt to delete the path `/` from the server `localhost`, the URL to delete would be `http://localhost:80/?`. 
By passing the -a or -A options to htcacheclean, the URLs within the cache will be listed as they are found, one URL per line. The -A option dumps the full cache entry after the URL, with fields in the following order:

**url**
The URL of the entry.

**header size**
The size of the header in bytes.

**body size**
The size of the body in bytes.

**status**
Status of the cached response.

**entity version**
The number of times this entry has been revalidated without being deleted.

**date**
Date of the response.

**expiry**
Expiry date of the response.

**request time**
Time of the start of the request.

**response time**
Time of the end of the request.

**body present**
If 0, no body is stored with this request, 1 otherwise.

**head request**
If 1, the entry contains a cached HEAD request with no body, 0 otherwise.
htcacheclean returns a zero status ("true") if all operations were successful, 1 otherwise. If an URL is specified, and the URL was cached and successfully removed, 0 is returned, 2 otherwise. If an error occurred during URL removal, 1 is returned.
htdbm - Manipulate DBM password databases

htdbm is used to manipulate the DBM format files used to store usernames and password for basic authentication of HTTP users via mod_authn_dbm. See the dbmmanage documentation for more information about these DBM files.

See also

httpd
dbmmanage
mod_authn_dbm
htdbm [ -TDBTYPE ] [ -i ] [ -c ] [ -m | -B | -d | -s | -p ] [ -C cost ] [ -t ] [ -v ] filename
username

htdbm -b [ -TDBTYPE ] [ -c ] [ -m | -B | -d | -s | -p ] [ -C cost ] [ -t ] [ -v ] filename
username password

htdbm -n [ -i ] [ -c ] [ -m | -B | -d | -s | -p ] [ -C cost ] [ -t ] [ -v ] username

htdbm -nb [ -c ] [ -m | -B | -d | -s | -p ] [ -C cost ] [ -t ] [ -v ] username password

htdbm -v [ -TDBTYPE ] [ -i ] [ -c ] [ -m | -B | -d | -s | -p ] [ -C cost ] [ -t ] [ -v ] filename
username

htdbm -vb [ -TDBTYPE ] [ -c ] [ -m | -B | -d | -s | -p ] [ -C cost ] [ -t ] [ -v ] filename
username password

htdbm -x [ -TDBTYPE ] filename username

htdbm -l [ -TDBTYPE ]
-b
Use batch mode; *i.e.*, get the password from the command line rather than prompting for it. This option should be used with extreme care, since the password is clearly visible on the command line. For script use see the -i option.

-i
Read the password from stdin without verification (for script usage).

-c
Create the passwdfile. If passwdfile already exists, it is rewritten and truncated. This option cannot be combined with the -n option.

-n
Display the results on standard output rather than updating a database. This option changes the syntax of the command line, since the passwdfile argument (usually the first one) is omitted. It cannot be combined with the -c option.

-m
Use MD5 encryption for passwords. On Windows and Netware, this is the default.

-B
Use bcrypt encryption for passwords. This is currently considered to be very secure.

-C
This flag is only allowed in combination with -B (bcrypt encryption). It sets the computing time used for the bcrypt algorithm (higher is more secure but slower, default: 5, valid: 4 to 31).

-d
Use crypt() encryption for passwords. The default on all
platforms but Windows and Netware. Though possibly supported by htdbm on all platforms, it is not supported by the httpd server on Windows and Netware. This algorithm is insecure by today's standards.

-s
Use SHA encryption for passwords. Facilitates migration from/to Netscape servers using the LDAP Directory Interchange Format (ldif). This algorithm is insecure by today's standards.

-p
Use plaintext passwords. Though htdbm will support creation on all platforms, the httpd daemon will only accept plain text passwords on Windows and Netware.

-l
Print each of the usernames and comments from the database on stdout.

-v
Verify the username and password. The program will print a message indicating whether the supplied password is valid. If the password is invalid, the program exits with error code 3.

-x
Delete user. If the username exists in the specified DBM file, it will be deleted.

-t
Interpret the final parameter as a comment. When this option is specified, an additional string can be appended to the command line; this string will be stored in the "Comment" field of the database, associated with the specified username.

filename
The filename of the DBM format file. Usually without the extension .db, .pag, or .dir. If -c is given, the DBM file is
created if it does not already exist, or updated if it does exist.

**username**
The username to create or update in *passwdfile*. If *username* does not exist in this file, an entry is added. If it does exist, the password is changed.

**password**
The plaintext password to be encrypted and stored in the DBM file. Used only with the `-b` flag.

**-TDBTYPE**
Type of DBM file (SDBM, GDBM, DB, or "default").
One should be aware that there are a number of different DBM file formats in existence, and with all likelihood, libraries for more than one format may exist on your system. The three primary examples are SDBM, NDBM, GNU GDBM, and Berkeley/Sleepycat DB 2/3/4. Unfortunately, all these libraries use different file formats, and you must make sure that the file format used by filename is the same format that htdbm expects to see. htdbm currently has no way of determining what type of DBM file it is looking at. If used against the wrong format, will simply return nothing, or may create a different DBM file with a different name, or at worst, it may corrupt the DBM file if you were attempting to write to it.

One can usually use the file program supplied with most Unix systems to see what format a DBM file is in.
htdbm returns a zero status ("true") if the username and password have been successfully added or updated in the DBM File. htdbm returns 1 if it encounters some problem accessing files, 2 if there was a syntax problem with the command line, 3 if the password was entered interactively and the verification entry didn't match, 4 if its operation was interrupted, 5 if a value is too long (username, filename, password, or final computed record), 6 if the username contains illegal characters (see the Restrictions section), and 7 if the file is not a valid DBM password file.
htdbm /usr/local/etc/apache/.htdbm-users jsmith

Adds or modifies the password for user jsmith. The user is prompted for the password. If executed on a Windows system, the password will be encrypted using the modified Apache MD5 algorithm; otherwise, the system's crypt() routine will be used. If the file does not exist, htdbm will do nothing except return an error.

htdbm -c /home/doe/public_html/.htdbm jane

Creates a new file and stores a record in it for user jane. The user is prompted for the password. If the file exists and cannot be read, or cannot be written, it is not altered and htdbm will display a message and return an error status.

htdbm -mb /usr/web/.htdbm-all jones Pwd4Steve

Encrypts the password from the command line (Pwd4Steve) using the MD5 algorithm, and stores it in the specified file.
Web password files such as those managed by htdbm should not be within the Web server's URI space -- that is, they should not be fetchable with a browser.

The use of the -b option is discouraged, since when it is used the unencrypted password appears on the command line.

When using the crypt() algorithm, note that only the first 8 characters of the password are used to form the password. If the supplied password is longer, the extra characters will be silently discarded.

The SHA encryption format does not use salting: for a given password, there is only one encrypted representation. The crypt() and MD5 formats permute the representation by prepending a random salt string, to make dictionary attacks against the passwords more difficult.

The SHA and crypt() formats are insecure by today's standards.
On the Windows platform, passwords encrypted with `htdbm` are limited to no more than 255 characters in length. Longer passwords will be truncated to 255 characters.

The MD5 algorithm used by `htdbm` is specific to the Apache software; passwords encrypted using it will not be usable with other Web servers.

Usernames are limited to 255 bytes and may not include the character `:`.
htdigest is used to create and update the flat-files used to store usernames, realm and password for digest authentication of HTTP users. Resources available from the Apache HTTP server can be restricted to just the users listed in the files created by htdigest.

This manual page only lists the command line arguments. For details of the directives necessary to configure digest authentication in httpd see the Apache manual, which is part of the Apache distribution or can be found at http://httpd.apache.org/.

See also

httpd
mod_auth_digest
htdigest [ -c ] passwdfile realm username
-c
  Create the passwdfile. If passwdfile already exists, it is deleted first.

passwdfile
  Name of the file to contain the username, realm and password. If -c is given, this file is created if it does not already exist, or deleted and recreated if it does exist.

realm
  The realm name to which the user name belongs. See http://tools.ietf.org/html/rfc2617#section-3.2.1 for more details.

username
  The user name to create or update in passwdfile. If username does not exist is this file, an entry is added. If it does exist, the password is changed.
This program is not safe as a setuid executable. Do *not* make it setuid.
htpasswd - Manage user files for basic authentication

htpasswd is used to create and update the flat-files used to store usernames and password for basic authentication of HTTP users. If htpasswd cannot access a file, such as not being able to write to the output file or not being able to read the file in order to update it, it returns an error status and makes no changes.

Resources available from the Apache HTTP server can be restricted to just the users listed in the files created by htpasswd. This program can only manage usernames and passwords stored in a flat-file. It can encrypt and display password information for use in other types of data stores, though. To use a DBM database see dbmmanage or htdbm.

htpasswd encrypts passwords using either bcrypt, a version of MD5 modified for Apache, SHA1, or the system's crypt() routine. Files managed by htpasswd may contain a mixture of different encoding types of passwords; some user records may have bcrypt or MD5-encrypted passwords while others in the same file may have passwords encrypted with crypt().

This manual page only lists the command line arguments. For details of the directives necessary to configure user authentication in httpd see the Apache manual, which is part of the Apache distribution or can be found at http://httpd.apache.org/.
See also

httpd
htdbm

The scripts in support/SHA1 which come with the distribution.
htpasswd [ -c ] [ -i ] [ -m | -B | -d | -s | -p ]
[ -C cost ] [ -D ] [ -v ] passwdfile username

htpasswd -b [ -c ] [ -m | -B | -d | -s | -p ] [ -C cost ] [ -D ] [ -v ] passwdfile username
password

htpasswd -n [ -i ] [ -m | -B | -d | -s | -p ] [ -C cost ] username

htpasswd -nb [ -m | -B | -d | -s | -p ] [ -C cost ] username password
-b
Use batch mode; *i.e.*, get the password from the command line rather than prompting for it. This option should be used with extreme care, since the password is clearly visible on the command line. For script use see the -i option. Available in 2.4.4 and later.

-i
Read the password from stdin without verification (for script usage).

-c
Create the passwdfile. If passwdfile already exists, it is rewritten and truncated. This option cannot be combined with the -n option.

-n
Display the results on standard output rather than updating a file. This is useful for generating password records acceptable to Apache for inclusion in non-text data stores. This option changes the syntax of the command line, since the passwdfile argument (usually the first one) is omitted. It cannot be combined with the -c option.

-m
Use MD5 encryption for passwords. This is the default (since version 2.2.18).

-B
Use bcrypt encryption for passwords. This is currently considered to be very secure.

-C
This flag is only allowed in combination with -B (bcrypt encryption). It sets the computing time used for the bcrypt algorithm (higher is more secure but slower, default: 5, valid: 4
Use Crypt() encryption for passwords. This is not supported by the httpd server on Windows and Netware. This algorithm limits the password length to 8 characters. This algorithm is **insecure** by today's standards. It used to be the default algorithm until version 2.2.17.

**-s**
Use SHA encryption for passwords. Facilitates migration from/to Netscape servers using the LDAP Directory Interchange Format (ldif). This algorithm is **insecure** by today's standards.

**-p**
Use plaintext passwords. Though htpasswd will support creation on all platforms, the httpd daemon will only accept plain text passwords on Windows and Netware.

**-D**
Delete user. If the username exists in the specified htpasswd file, it will be deleted.

**-v**
Verify password. Verify that the given password matches the password of the user stored in the specified htpasswd file. Available in 2.4.5 and later.

**passwdfile**
Name of the file to contain the user name and password. If -c is given, this file is created if it does not already exist, or rewritten and truncated if it does exist.

**username**
The username to create or update in passwdfile. If username does not exist in this file, an entry is added. If it does exist, the password is changed.
password

The plaintext password to be encrypted and stored in the file. Only used with the -b flag.
htpasswd returns a zero status ("true") if the username and password have been successfully added or updated in the passwdfile. htpasswd returns 1 if it encounters some problem accessing files, 2 if there was a syntax problem with the command line, 3 if the password was entered interactively and the verification entry didn’t match, 4 if its operation was interrupted, 5 if a value is too long (username, filename, password, or final computed record), 6 if the username contains illegal characters (see the Restrictions section), and 7 if the file is not a valid password file.
htpasswd /usr/local/etc/apache/.htpasswd-users jsmith

Adds or modifies the password for user jsmith. The user is prompted for the password. The password will be encrypted using the modified Apache MD5 algorithm. If the file does not exist, htpasswd will do nothing except return an error.

htpasswd -c /home/doe/public_html/.htpasswd jane

Creates a new file and stores a record in it for user jane. The user is prompted for the password. If the file exists and cannot be read, or cannot be written, it is not altered and htpasswd will display a message and return an error status.

htpasswd -db /usr/web/.htpasswd-all jones Pwd4Steve

Encrypts the password from the command line (Pwd4Steve) using the crypt( ) algorithm, and stores it in the specified file.
Web password files such as those managed by `htpasswd` should *not* be within the Web server's URI space -- that is, they should not be fetchable with a browser.

This program is not safe as a setuid executable. Do *not* make it setuid.

The use of the `-b` option is discouraged, since when it is used the unencrypted password appears on the command line.

When using the `crypt()` algorithm, note that only the first 8 characters of the password are used to form the password. If the supplied password is longer, the extra characters will be silently discarded.

The SHA encryption format does not use salting: for a given password, there is only one encrypted representation. The `crypt()` and MD5 formats permute the representation by prepending a random salt string, to make dictionary attacks against the passwords more difficult.

The SHA and `crypt()` formats are insecure by today's standards.
On the Windows platform, passwords encrypted with `htpasswd` are limited to no more than 255 characters in length. Longer passwords will be truncated to 255 characters.

The MD5 algorithm used by `htpasswd` is specific to the Apache software; passwords encrypted using it will not be usable with other Web servers.

Usernames are limited to 255 bytes and may not include the character `:`.
logresolve - Resolve IP-addresses to hostnames in Apache log files

logresolve is a post-processing program to resolve IP-addresses in Apache's access logfiles. To minimize impact on your nameserver, logresolve has its very own internal hash-table cache. This means that each IP number will only be looked up the first time it is found in the log file.

Takes an Apache log file on standard input. The IP addresses must be the first thing on each line and must be separated from the remainder of the line by a space.
logresolve [ -s filename ] [ -c ] < access_log > access_log.new
-s *filename*

Specifies a filename to record statistics.

- c

This causes `log resolve` to apply some DNS checks: after finding the hostname from the IP address, it looks up the IP addresses for the hostname and checks that one of these matches the original address.
rotatelogs - Piped logging program to rotate Apache logs

rotatelogs is a simple program for use in conjunction with Apache's piped logfile feature. It supports rotation based on a time interval or maximum size of the log.
rotatelogs [ -l ] [ -L linkname ] [ -p program ] [ -f ] [ -t ] [ -v ] [ -e ] [ -c ] [ -n number-of-files ] logfile rotationtime|filesize(B|K|M|G) [ offset ]
-l
Causes the use of local time rather than GMT as the base for the interval or for strftime(3) formatting with size-based rotation.

-L linkname
Causes a hard link to be made from the current logfile to the specified link name. This can be used to watch the log continuously across rotations using a command like tail -F linkname.

-p program
If given, rotatelogs will execute the specified program every time a new log file is opened. The filename of the newly opened file is passed as the first argument to the program. If executing after a rotation, the old log file is passed as the second argument. rotatelogs does not wait for the specified program to terminate before continuing to operate, and will not log any error code returned on termination. The spawned program uses the same stdin, stdout, and stderr as rotatelogs itself, and also inherits the environment.

-f
Causes the logfile to be opened immediately, as soon as rotatelogs starts, instead of waiting for the first logfile entry to be read (for non-busy sites, there may be a substantial delay between when the server is started and when the first request is handled, meaning that the associated logfile does not "exist" until then, which causes problems from some automated logging tools)

-t
Causes the logfile to be truncated instead of rotated. This is useful when a log is processed in real time by a command like tail, and there is no need for archived data. No suffix will be
added to the filename, however format strings containing '%' characters will be respected.

- \v
  Produce verbose output on STDERR. The output contains the result of the configuration parsing, and all file open and close actions.

- \e
  Echo logs through to stdout. Useful when logs need to be further processed in real time by a further tool in the chain.

- \c
  Create log file for each interval, even if empty.

- \n \textit{number-of-files}
  Use a circular list of filenames without timestamps. With -n 3, the series of log files opened would be "logfile", "logfile.1", "logfile.2", then overwriting "logfile". Available in 2.4.5 and later.

\textit{logfile}
  The path plus basename of the logfile. If \textit{logfile} includes any '%' characters, it is treated as a format string for \texttt{strftime(3)}. Otherwise, the suffix \texttt{.nnnnnnnnnn} is automatically added and is the time in seconds (unless the -t option is used). Both formats compute the start time from the beginning of the current period. For example, if a rotation time of 86400 is specified, the hour, minute, and second fields created from the \texttt{strftime(3)} format will all be zero, referring to the beginning of the current 24-hour period (midnight).

When using \texttt{strftime(3)} filename formatting, be sure the log file format has enough granularity to produce a different file name each time the logs are rotated. Otherwise rotation will overwrite the same file instead of starting a new one. For
example, if logfile was /var/log/errorlog.%Y-%m-%d
with log rotation at 5 megabytes, but 5 megabytes was
reached twice in the same day, the same log file name would
be produced and log rotation would keep writing to the same
file.

**rotationtime**

The time between log file rotations in seconds. The rotation
occurs at the beginning of this interval. For example, if the
rotation time is 3600, the log file will be rotated at the
beginning of every hour; if the rotation time is 86400, the log
file will be rotated every night at midnight. (If no data is logged
during an interval, no file will be created.)

**filesize(B|K|M|G)**

The maximum file size in followed by exactly one of the letters
B (Bytes), K (KBytes), M (MBytes) or G (GBytes).

When time and size are specified, the size must be given after
the time. Rotation will occur whenever either time or size
limits are reached.

**offset**

The number of minutes offset from UTC. If omitted, zero is
assumed and UTC is used. For example, to use local time in
the zone UTC -5 hours, specify a value of -300 for this
argument. In most cases, -1 should be used instead of
specifying an offset.
CustomLog "|bin/rotatelogs /var/log/logfile 86400" common

This creates the files /var/log/logfile.nnnn where nnnn is the system time at which the log nominally starts (this time will always be a multiple of the rotation time, so you can synchronize cron scripts with it). At the end of each rotation time (here after 24 hours) a new log is started.

CustomLog "|bin/rotatelogs -l /var/log/logfile.%Y.%m.%d 86400" common

This creates the files /var/log/logfile.yyyy.mm.dd where yyyy is the year, mm is the month, and dd is the day of the month. Logging will switch to a new file every day at midnight, local time.

CustomLog "|bin/rotatelogs /var/log/logfile 5M" common

This configuration will rotate the logfile whenever it reaches a size of 5 megabytes.

ErrorLog "|bin/rotatelogs /var/log/errorlog.%Y-%m-%d-%H_%M_%S 5M"

This configuration will rotate the error logfile whenever it reaches a size of 5 megabytes, and the suffix to the logfile name will be created of the form errorlog.YYYY-mm-dd-HH_MM_SS.

CustomLog "|bin/rotatelogs -t /var/log/logfile 86400" common

This creates the file /var/log/logfile, truncating the file at startup and then truncating the file once per day. It is expected in this scenario that a separate process (such as tail) would process the file in real time.
The following logfile format string substitutions should be supported by all strftime(3) implementations, see the strftime(3) man page for library-specific extensions.

<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%A</td>
<td>full weekday name (localized)</td>
</tr>
<tr>
<td>%a</td>
<td>3-character weekday name (localized)</td>
</tr>
<tr>
<td>%B</td>
<td>full month name (localized)</td>
</tr>
<tr>
<td>%b</td>
<td>3-character month name (localized)</td>
</tr>
<tr>
<td>%c</td>
<td>date and time (localized)</td>
</tr>
<tr>
<td>%d</td>
<td>2-digit day of month</td>
</tr>
<tr>
<td>%H</td>
<td>2-digit hour (24 hour clock)</td>
</tr>
<tr>
<td>%I</td>
<td>2-digit hour (12 hour clock)</td>
</tr>
<tr>
<td>%j</td>
<td>3-digit day of year</td>
</tr>
<tr>
<td>%M</td>
<td>2-digit minute</td>
</tr>
<tr>
<td>%m</td>
<td>2-digit month</td>
</tr>
<tr>
<td>%p</td>
<td>am/pm of 12 hour clock (localized)</td>
</tr>
<tr>
<td>%S</td>
<td>2-digit second</td>
</tr>
<tr>
<td>%U</td>
<td>2-digit week of year (Sunday first day of week)</td>
</tr>
<tr>
<td>%w</td>
<td>2-digit week of year (Monday first day of week)</td>
</tr>
<tr>
<td>%W</td>
<td>2-digit week of year (Sunday first day of week)</td>
</tr>
<tr>
<td>%x</td>
<td>date (localized)</td>
</tr>
<tr>
<td>%y</td>
<td>4-digit year</td>
</tr>
<tr>
<td>%Y</td>
<td>2-digit year</td>
</tr>
<tr>
<td>%z</td>
<td>time zone name</td>
</tr>
<tr>
<td>%%</td>
<td>literal `%'</td>
</tr>
</tbody>
</table>
Other Programs

This page used to contain documentation for programs which now have their own docs pages. Please update any links.

log_server_status

split-logfile
Relevant Standards

This page documents all the relevant standards that the Apache HTTP Server follows, along with brief descriptions.

In addition to the information listed below, the following resources should be consulted:

- [http://purl.org/NET/http-errata](http://purl.org/NET/http-errata) - HTTP/1.1 Specification Errata

Notice

This document is not yet complete.
Regardless of what modules are compiled and used, Apache as a basic web server complies with the following IETF recommendations:

**RFC 1945 (Informational)**

The Hypertext Transfer Protocol (HTTP) is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia information systems. This documents HTTP/1.0.

**RFC 2616 (Standards Track)**

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. This documents HTTP/1.1.

**RFC 2396 (Standards Track)**

A Uniform Resource Identifier (URI) is a compact string of characters for identifying an abstract or physical resource.

**RFC 4346 (Standards Track)**

The TLS protocol provides communications security over the Internet. It provides encryption, and is designed to prevent eavesdropping, tampering, and message forgery.
Regarding the Hypertext Markup Language, Apache complies with the following IETF and W3C recommendations:

**RFC 2854 (Informational)**
This document summarizes the history of HTML development, and defines the "text/html" MIME type by pointing to the relevant W3C recommendations.

**HTML 4.01 Specification (Errata)**
This specification defines the HyperText Markup Language (HTML), the publishing language of the World Wide Web. This specification defines HTML 4.01, which is a subversion of HTML 4.

**HTML 3.2 Reference Specification**
The HyperText Markup Language (HTML) is a simple markup language used to create hypertext documents that are portable from one platform to another. HTML documents are SGML documents.

**XHTML 1.1 - Module-based XHTML (Errata)**
This Recommendation defines a new XHTML document type that is based upon the module framework and modules defined in Modularization of XHTML.

**XHTML 1.0 The Extensible HyperText Markup Language (Second Edition) (Errata)**
This specification defines the Second Edition of XHTML 1.0, a reformulation of HTML 4 as an XML 1.0 application, and three DTDs corresponding to the ones defined by HTML 4.
Concerning the different methods of authentication, Apache follows the following IETF recommendations:

**RFC 2617** *(Standards Track)*  
"HTTP/1.0", includes the specification for a Basic Access Authentication scheme.
The following links document ISO and other language and country code information:

**ISO 639-2**
ISO 639 provides two sets of language codes, one as a two-letter code set (639-1) and another as a three-letter code set (this part of ISO 639) for the representation of names of languages.

**ISO 3166-1**
These pages document the country names (official short names in English) in alphabetical order as given in ISO 3166-1 and the corresponding ISO 3166-1-alpha-2 code elements.

**BCP 47 (Best Current Practice), RFC 3066**
This document describes a language tag for use in cases where it is desired to indicate the language used in an information object, how to register values for use in this language tag, and a construct for matching such language tags.

**RFC 3282 (Standards Track)**
This document defines a "Content-language:" header, for use in cases where one desires to indicate the language of something that has RFC 822-like headers, like MIME body parts or Web documents, and an "Accept-Language:" header for use in cases where one wishes to indicate one's preferences with regard to language.
Apache

Apache
Apache

**MPM**
  "MPM" Apache MPM

**Base**
  "Base"

**Extension**
  "Extension"

**Experimental**
  "Experimental" Apache

**External**
  "External" Apache ("")
LoadModule
Apache
"..."
Apache
<VirtualHost> httpd.conf, srm.conf, access.conf

<Directory> .htaccess
</Directory>
</VirtualHost>

<VirtualHost>

<Directory>, <Location>, <Files:

.htaccess

.httpd.conf .htaccess
</VirtualHost>

(: Boolean OR

<Direct>
.htaccess

AllowOverride ()
Apache

Core
  "Core"  Apache

MPM
  "MPM"

Base
  "Base"

Extension
  "Extension"  Apache

Experimental
  "Experimental"  Apache
Apache

- Apache HTTP
- Core
Listen
   AcceptFilter *protocol* accept_filter

Core
   core
   2.1.5

Listen OS
   Accept Filter
   Linux
   TCP_DEFER_ACCEPT

FreeBSD:
   AcceptFilter http httpready
   AcceptFilter https dataready

httpready Accept Filter HTTP HTTPS
   accf_data(9)

Linux:
   AcceptFilter http data
   AcceptFilter https data

Linux    TCP_DEFER_ACCEPT HTTP TCP_DEFER_ACCEPT Linux
   man
   none Accept Filter
   AcceptFilter nntp none
AcceptPathInfo On|Off|Default
AcceptPathInfo Default

..., .htaccess
FileInfo
Core
core
Apache 2.0.30

(test/  here.html
/test/nothere.html/more  /more P)

AcceptPathInfo : 
Off
On
Default

PATH_INFO

AcceptPathInfo PATH_INFO PATH_INFO

<Files "mypaths.shtml">
  Options +Includes
  SetOutputFilter INCLUDES
  AcceptPathInfo On
AccessFileName filename [filename] ...
AccessFileName .htaccess

<Directory />
  AllowOverride None
</Directory>

- AllowOverride
- .htaccess
<table>
<thead>
<tr>
<th>AddDefaultCharset</th>
<th>On</th>
<th>Off</th>
<th>charset</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddDefaultCharset</td>
<td>Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.,.,.htaccess</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FileInfo</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>core</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- AddDefaultCharset

- AddCharset
AllowEncodedSlashes On|Off
AllowEncodedSlashes Off

Core
core
Apache 2.0.46

AllowEncodedSlashes ( / %:
URL 404 (Not found)

AllowEncodedSlashes On

%2F ()
%5C URL

- AcceptPathInfo
<table>
<thead>
<tr>
<th>directive-type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
<td>Allows all directives in the .htaccess file.</td>
</tr>
<tr>
<td>None</td>
<td>Disables all directives in the .htaccess file.</td>
</tr>
</tbody>
</table>

**AuthConfig**

- AuthDBMGroupFile
- AuthGroupFile
- AuthName
- AuthType
- AuthUserFile
- Require

**FileInfo**

- Default
- LanguagePriority
- SetHandler
- SetInputFilter
- SetOutputFilter
- mod_mime
- Add*
- Remove*
- Header
- RequestHeader
- SetEnvIf
- SetEnvIfNoCase
- BrowserMatch
- CookieExpires
- CookieDomain
- CookieStyle
- CookieTracking
- CookieName
mod_rewrite  RewriteEngine, RewriteOptions, RewriteBase, RewriteCond, RewriteRule
mod_actions  Action

Indexes

AddIconByType, DefaultIcon, DirectoryIndex, FancyIndexing, HeaderName, IndexIgnore, IndexOptions, ReadmeName

Limit

(Allow

Options[=Option, ...]

:

AllowOverride AuthConfig Indexes

AuthConfig  Indexes

• AccessFileName
•
• .htaccess
Individual directives that are allowed in .htaccess files

AllowOverrideList None|directive [directive-type] ...

AllowOverrideList None

Core

core

- AccessFileName
- AllowOverride
- Configuration Files
- .htaccess Files
CGI

CGIMapExtension *cgi-path* .extension

, .htaccess

FileInfo

Core

core

NetWare

Apache CGI

.foo .foo CGI FOO
CGIPassAuth

Enables passing HTTP authorization headers to scripts as CGI variables

CGIPassAuth 0n|0ff
CGIPassAuth Off

.htaccess
AuthConfig
Core
core

Available in Apache HTTP Server 2.4.13 and later
CGIVar: Controls how some CGI variables are set
     variable rule
     .htaccess
     FileInfo
     Core
     core
     Available in Apache HTTP Server 2.4.21 and later
<table>
<thead>
<tr>
<th></th>
<th>Content-MD5 HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>ContentDigest On</td>
</tr>
<tr>
<td></td>
<td>ContentDigest Off</td>
</tr>
<tr>
<td></td>
<td>.htaccess</td>
</tr>
<tr>
<td></td>
<td>Options</td>
</tr>
<tr>
<td></td>
<td>Core</td>
</tr>
<tr>
<td></td>
<td>core</td>
</tr>
</tbody>
</table>

RFC1864 RFC2616

Content-MD5

Content-MD5: AuLb7Dp1rqTrtxz2m9kRpa==
<table>
<thead>
<tr>
<th><strong>DefaultRuntimeDir</strong></th>
<th><em>directory-path</em></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DefaultRuntimeDir</strong></td>
<td><strong>DEFAULT_REL_RUNTIMEDIR</strong> (logs/)</td>
</tr>
</tbody>
</table>

- **Core**
- **core**

- Available in Apache 2.4.2 and later

- [the security tips](#) for information on how to properly set permissions on the **ServerRoot**
<table>
<thead>
<tr>
<th>MIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>DefaultType MIME-type</td>
</tr>
<tr>
<td>DefaultType text/plain</td>
</tr>
<tr>
<td>.htaccess</td>
</tr>
<tr>
<td>FileInfo</td>
</tr>
<tr>
<td>Core</td>
</tr>
<tr>
<td>core</td>
</tr>
<tr>
<td>none Apache 2.2.7</td>
</tr>
</tbody>
</table>

**DefaultType image/gif**

```
.gif
```

**DefaultType None**

```
DefaultType None httpd-2.2.7
```

**ForceType MIME**
Define parameter-name

Core

core

httpd -D
<Directory directory-path> ... </Directory>

Core

core

path Unix

/home/user/public_html <Directory
/ /*/public_html> <Directory
/home/*/public_html>

<Directory /usr/local/httpd/htdocs>
  Options Indexes FollowSymLinks
</Directory>

<Directory /~^
          "^/www/.*/[0-9]{3}">
</Directory>

<Directory />
  AllowOverride None
</Directory>

<Directory /home/>
  AllowOverride FileInfo
</Directory>
AllowOverride None (.htaccess)
AllowOverride FileInfo (/home)
/home/.htaccess, /home/web/.htaccess, /home/web/dir/.htaccess

<Directory ~ abc$>
  # ... directives here ...
</Directory>

<Directory> .htaccess /home/abc/public_html/abc <Directory>

Apache  <Directory />  Allow from All
URL Apache

<Directory />
  Order Deny,Allow
  Deny from All
</Directory>

httpd.conf  <Directory>
<LimitExcept>

•

"
<DirectoryMatch regex> ...
</DirectoryMatch>

Core
core

<Directory>

<DirectoryMatch "^/www/(.+)?[0-9]{3}"

/www/ 3

- <Directory> <Directory>
- 

-
DocumentRoot directory-path

DocumentRoot /usr/local/apache/htdocs

Core

core

httpd

Alias

DocumentRoot /usr/web

http://www.my.host.com/index.html

/usr/web/index.html
directory-path

DocumentRoot

• URL
Contains directives that apply only if the condition of a previous `<If>` or `<ElseIf>` section is not satisfied by a request at runtime:

```
<Else> ... </Else>
```

All .htaccess

Core

core

Nested conditions are evaluated in 2.4.26 and later

- `<If>`
- `<ElseIf>`
- How `<Directory>`, `<Location>`, `<Files>` sections work for an explanation of how these different sections are combined when a request is received. `<If>`, `<ElseIf>`, and `<Else>` are applied last.
Contains directives that apply only if a condition is satisfied by a request at runtime while the condition of a previous `<If>` or `<ElseIf>` section is not satisfied:

```html
<ElseIf expression> ... </ElseIf>
```

All core nested conditions are evaluated in 2.4.26 and later.

- **Expressions in Apache HTTP Server**, for a complete reference and more examples.
- `<If>`
- `<Else>`
- **How `<Directory>, `<Location>, `<Files> sections work** for an explanation of how these different sections are combined when a request is received. `<If>, `<ElseIf>, and `<Else> are applied last.
EnableMMAP On|Off
EnableMMAP On

NFS DocumentRoot httpd

<NFS DocumentRoot httpd
<Directory "/path-to-nfs-files">
    EnableMMAP Off
</Directory>
sendfile read send

- sendfile
- Linux sendfile
- Itanium
- DocumentRoot (NFS SMB)
- EnableSendfile 0n|0ff
- EnableSendfile 0n
- ... .htaccess
- FileInfo
- Core
- core
- 2.0.44

httpd

EnableSendfile Off

NFS SMB

```
<Directory "/path-to-nfs-files">
  EnableSendfile Off
</Directory>
```
Abort configuration parsing with a custom error message

Error message

., .htaccess

Core
core

2.3.9 and later
ErrorDocument error-code document

FileInfo

Core
core

Apache

1. Apache

2. 

3. URL-path

4. URL

2 4 ErrorDocument Apache

URL (/) web-path ( DocumentRoot :

ErrorDocument 500 http://foo.example.com/cgi-bin/tester
ErrorDocument 404 /cgi-bin/bad_urls.pl
ErrorDocument 401 /subscription_info.html
ErrorDocument 403 "Sorry can't allow you access today"

default Apache

ErrorDocument Apache

ErrorDocument 404 /cgi-bin/bad_urls.pl

<Directory /web/docs>
  ErrorDocument 404 default
</Directory>
URL ( http ) ErrorDocument

Apache

ErrorDocument 401 URL 401

ErrorDocument 401

Microsoft Internet Explorer (MSIE)

Error

2.0
ErrorLog

ErrorLog file-path|syslog[:facility]

ErrorLog logs/error_log (Unix) ErrorLog logs/error.log (Windows and OS/2)

Core

core

ErrorLog

ErrorLog /var/log/httpd/error_log

file-path ()

ErrorLog "|/usr/local/bin/httpd_errors"

syslog syslogd(8)
syslog:facility syslog(1)

ErrorLog syslog:user

Unix
- LogLevel
- Apache
Format specification for error log entries

**ErrorLogFormat** [connection|request] *format*

- Core
- core

- **ErrorLog**
- **LogLevel**
- **Apache HTTP Server Log Files**
Keep track of extended status information for each request

ExtendedStatus On|Off

ExtendedStatus Off[*]

Core
core
<table>
<thead>
<tr>
<th>FileETag</th>
<th>ETag HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileETag</td>
<td>component</td>
</tr>
<tr>
<td>FileETag</td>
<td>INode MTime Size</td>
</tr>
<tr>
<td>, , , .htaccess</td>
<td>FileInfo</td>
</tr>
<tr>
<td></td>
<td>Core</td>
</tr>
<tr>
<td></td>
<td>core</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FileETag</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETag inode, (mtime)</td>
</tr>
</tbody>
</table>

| INode |
|inode |

| MTime |

| Size |

| All |

| FileETag INode MTime Size |

| None |
| ETag |

| INode, MTime, Size + - |

<p>| FileETag INode MTime Size |
| ( | ) |</p>
<table>
<thead>
<tr>
<th>WebDAV</th>
<th>mod_dav_fs</th>
<th>Directory</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_dav_fs</td>
<td></td>
<td>INode MTime Size</td>
</tr>
<tr>
<td>ETag</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
<Files filename> ...
</Files>

..., .htaccess

All
Core
core

<Location>
<Directory>
filename
</Directory>
<Files ~ ".(gif|jpe?g|png)"/>

<Directory> <Location> <Files> .
<FilesMatch regex> ... </FilesMatch>
...
.htaccess
All
Core
core

<FilesMatch>  
<Files>

<FilesMatch "\.\.(gif|jpe?g|png)$"
### Core

```
<Directory>
<Location>
<Files>
MIME-type image/gif
</Files>
</Location>
# force all files to be image/gif:
<Location /images>
  ForceType image/gif
</Location>
# but normal mime-type associations here:
<Location /images/mixed>
  ForceType None
</Location>
```
GprofDir: Directory to write gmon.out profiling data to.
GprofDir /tmp/gprof/|/tmp/gprof/%

Core
core
<table>
<thead>
<tr>
<th>DNS</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_authz_host</td>
<td>2</td>
</tr>
<tr>
<td>Double</td>
<td>2</td>
</tr>
<tr>
<td>REMOTE_HOST</td>
<td></td>
</tr>
<tr>
<td></td>
<td>bin</td>
</tr>
</tbody>
</table>
Modify restrictions on HTTP Request Messages

HttpProtocolOptions [Strict|Unsafe]
[RegisteredMethods|LenientMethods]
[Allow0.9|Require1.0]

HttpProtocolOptions Strict LenientMethods Allow0.9

Core

core

2.2.32 or 2.4.24 and later
<If expression> ... </If>
,
, .htaccess
All
Core
core

<If>

<If "$req{Host} = ''">

*Host: HTTP/1.0*

- `<Directory>`, `<Location>`, `<Files>`

</Files>
<IfDefine test>...

  test

  test

<IfDefine>

  parameter-name
  !parameter-name

<IfDefine>

  httpd

<IfDefine>

  httpd -DReverseProxy -DUseCache -DMemCache ...

# httpd.conf
<IfDefine ReverseProxy>
  LoadModule proxy_module modules/mod_proxy.so
  LoadModule proxy_http_module modules/mod_proxy_http.so
<IfDefine UseCache>
  LoadModule cache_module modules/mod_cache.so
  <IfDefine MemCache>
    LoadModule mem_cache_module modules/mod_mem_cache.so
  </IfDefine>
</IfDefine>
<IfDefine !MemCache>
  LoadModule cache_disk_module modules/mod_cache_disk.so
</IfDefine>
</IfDefine>
</IfDefine>
<IfModule test>...<IfModule>
  test
</IfModule>

<IfModule>
  test
</IfModule>

- module
- !module

module

<IfModule>

module

STANDARD20

<IfModule>
Include `file-path|directory-path`

Core `core` 2.0.41

```
ServerRoot

Include /usr/local/apache2/conf/ssl.conf
Include /usr/local/apache2/conf/vhosts/*.conf

ServerRoot:

Include conf/ssl.conf
Include conf/vhosts/* .conf
```

- `apachectl`
Includes other configuration files from within the server configuration files

**IncludeOptional** `file-path|directory-path|wildcard`

- ,
- Core
- core
- Available in 2.3.6 and later

- Include
- `apachectl`
<table>
<thead>
<tr>
<th>HTTP</th>
<th>KeepAlive On</th>
<th>Off</th>
<th>KeepAlive On</th>
<th>KeepAlive Off</th>
</tr>
</thead>
</table>

HTTP/1.0 Keep-Alive HTTP/1.1

<table>
<thead>
<tr>
<th>TCP</th>
<th>HTML 50%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CGI  SSI</td>
</tr>
</tbody>
</table>

HTTP/1.0 Keep-Alive HTTP/1.1

- MaxKeepAliveRequests
KeepAliveTimeout

KeepAliveTimeout seconds
KeepAliveTimeout 5

Core
core

Apache

KeepAliveTimeout

NameVirtualHost
<LimitExcept method [method] ... > ... 
</LimitExcept>

..., .htaccess

All

Core
core

<LimitExcept>
</LimitExcept>

HTTP

<Limit>

<LimitExcept POST GET>
    Require valid-user
</LimitExcept>
LimitInternalRecursion number [number]

LimitInternalRecursion 10,

Core
core

Apache 2.0.47

Action mod_dir DirectoryIndex

LimitInternalRecursion

LimitInternalRecursion 5
HTTP
LimitRequestBody bytes
LimitRequestBody 0
..., .htaccess
All
Core
core

bytes 0 () 2147483647 (2GB)

LimitRequestBody

100K

LimitRequestBody 102400
HTTP

LimitRequestFields number

LimitRequestFields 100

Core
core

number 0 () 32767
DEFAULT_LIMIT_REQUEST_FIELDS (100)

LimitRequestBody HTTP

HTTP

LimitRequestFields 50
HTTP

LimitRequestFieldSize bytes

LimitRequestFieldSize 8190

Core
core

LimitRequestFieldSize HTTP

LimitRequestFieldSize 4094
<table>
<thead>
<tr>
<th>HTTP</th>
<th>LimitRequestLine</th>
<th>bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LimitRequestLine</td>
<td>8190</td>
</tr>
<tr>
<td>Core</td>
<td></td>
<td></td>
</tr>
<tr>
<td>core</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HTTP</th>
<th>LimitRequestLine</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LimitRequestLine</td>
<td>4094</td>
</tr>
</tbody>
</table>


XML

LimitXMLRequestBody bytes
LimitXMLRequestBody 1000000

All
Core
core

XML ()

LimitXMLRequestBody 0
<Location> URL </Location>

<Files>

<Location>

<LocationMatch>
</Location>

() URL /path/ http scheme://servername/path

URL ? *
~

<Location ~ "/(extra|special)/data”>

URL /extra/data /special/data

<LocationMatch> <Location>

<Location> SetHandler
<Location /status>
    SetHandler server-status
    Order Deny,Allow
    Deny from all
    Allow from .example.com
</Location>

/() URL
    <LocationMatch>
        <Location>
            <LocationMatch ^/abc> /abc URL
        </Location>
    </LocationMatch>
    () <Location>
    <Location> proxy
        /abc//def
    </Location>
<table>
<thead>
<tr>
<th>URL</th>
<th>&lt;LocationMatch regex&gt; ... &lt;/LocationMatch&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>,</td>
<td>Core</td>
</tr>
<tr>
<td>core</td>
<td></td>
</tr>
</tbody>
</table>

<LocationMatch>  
<Location>  
URL

<LocationMatch "/(extra|special)/data">

URL  /extra/data  /special/data

•
<table>
<thead>
<tr>
<th>LogLevel</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>emerg</td>
<td>Child cannot open lock file. Exiting ()</td>
</tr>
<tr>
<td>alert</td>
<td>getpwuid: couldn't determine user name from uid (getpwuid: UID )</td>
</tr>
<tr>
<td>crit</td>
<td>socket: Failed to get a socket, exiting child (socket: )</td>
</tr>
<tr>
<td>error</td>
<td>Premature end of script headers ()</td>
</tr>
<tr>
<td>warn</td>
<td>child process 1234 did not exit, sending another SIGHUP ( 1234  SIGHUP )</td>
</tr>
<tr>
<td>notice</td>
<td>httpd: caught SIGBUS, attempting to dump core in ... (httpd: SIGBUS ... )</td>
</tr>
<tr>
<td>info</td>
<td>&quot;Server seems busy, (you may need to increase StartServers, or Min/MaxSpareServers)...&quot; ( (StartServers Min/MaxSpareServers ))</td>
</tr>
<tr>
<td>debug</td>
<td>&quot;Opening config file ...&quot; (...)</td>
</tr>
</tbody>
</table>
LogLevel notice

notice
<table>
<thead>
<tr>
<th>MaxKeepAliveRequests number</th>
<th>MaxKeepAliveRequests 100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core</td>
<td>core</td>
</tr>
</tbody>
</table>

MaxKeepAliveRequests 500

KeepAlive
MaxRangeOverlaps: Number of overlapping ranges (eg: 100-200, 150-300) allowed before returning the complete resource

MaxRangeOverlaps default | unlimited | none | number-of-ranges  
MaxRangeOverlaps 20

Core

core

Available in Apache HTTP Server 2.3.15 and later
MaxRangeReversals: Number of range reversals (eg: 100-200, 50-70) allowed before returning the complete resource. Default values are:
- MaxRangeReversals default | unlimited | none
- number-of-ranges
- MaxRangeReversals 20

Available in Apache HTTP Server 2.3.15 and later
MaxRanges

Number of ranges allowed before returning the complete resource

MaxRanges default | unlimited | none | number-of-ranges

MaxRanges 200

Core

Available in Apache HTTP Server 2.3.15 and later
<table>
<thead>
<tr>
<th>MergeTrailers</th>
<th>Determines whether trailers are merged into headers</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MergeTrailers [on</td>
</tr>
<tr>
<td></td>
<td>MergeTrailers off</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td></td>
</tr>
<tr>
<td>core</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2.4.11 and later</td>
</tr>
</tbody>
</table>
:: Configures mutex mechanism and lock file directory for all or specified mutexes
:: Mutex mechanism [default|mutex-name] ...
:: [OmitPID]
:: Mutex default
::
:: Core
::  core
:: Available in Apache HTTP Server 2.3.4 and later
<table>
<thead>
<tr>
<th>NameVirtualHost</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>111.22.33.44</td>
<td>IPv6</td>
</tr>
<tr>
<td>[2001:db8::a00:20ff:feaa7:ceca]:8080</td>
<td>*</td>
</tr>
<tr>
<td><em>default</em></td>
<td>NameVirtualHost</td>
</tr>
</tbody>
</table>

```
<VirtualHost>
  NameVirtualHost

  NameVirtualHost 1.2.3.4
  <VirtualHost 1.2.3.4>
    # ...
  </VirtualHost>
</VirtualHost>
Options

option  None  1

All
  MultiViews

ExecCGI
  mod_cgi  CGI

FollowSymLinks

<Directory>
  <Location>
      symlink
</Location>
  <Directory>

Includes
  mod_include  SSI

IncludesNOEXEC
  #exec  #exec  CGI
  virtual  ScriptAlias  CGI

Indexes
<Directory /web/docs>
  Options Indexes FollowSymLinks
</Directory>

<Directory /web/docs/spec>
  Options Includes
</Directory>

/web/docs/spec Includes

<Directory /web/docs>
  Options Indexes FollowSymLinks
</Directory>
<Directory /web/docs/spec>
  Options +Includes -Indexes
</Directory>

/web/docs/spec   FollowSymLinks   Includes

-IncludesNOEXEC   -Includes   SSI

All
<table>
<thead>
<tr>
<th>Protocol for a listening socket</th>
</tr>
</thead>
<tbody>
<tr>
<td>Protocol <strong>protocol</strong></td>
</tr>
<tr>
<td>Core</td>
</tr>
<tr>
<td>core</td>
</tr>
<tr>
<td>Available in Apache 2.1.5 and later. On Windows, from Apache 2.3.3 and later.</td>
</tr>
</tbody>
</table>

- **AcceptFilter**
- **Listen**
Protocols available for a server/virtual host:

- Protocols `protocol` ...
- Protocols `http/1.1`
- Core
- core

Only available from Apache 2.4.17 and later.

- **ProtocolsHonorOrder**
Determines if order of Protocols determines precedence during negotiation.

ProtocolsHonorOrder On|Off

ProtocolsHonorOrder On

, Core
core

Only available from Apache 2.4.17 and later.

- **Protocols**
QualifyRedirectURL

Controls whether the REDIRECT_URL environment variable is fully qualified.

QualifyRedirectURL ON|OFF

QualifyRedirectURL OFF

 FileInfo
 Core
core

Directive supported in 2.4.18 and later. 2.4.17 acted as if 'QualifyRedirectURL ON' was configured.
<table>
<thead>
<tr>
<th></th>
<th>Register non-standard HTTP methods</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>RegisterHttpMethod method [method [...]</code></td>
</tr>
</tbody>
</table>

```yaml
---
| Core |
| core |
```
RlimitCPU

Apache CPU
RlimitCPU seconds|max [seconds|max]
...
..., .htaccess
All
Core
core

Apache fork
Apache fork

CPU

- RLimitMEM
- RLimitNPROC
Apache

RLimitMEM bytes|max [bytes|max]

..., .htaccess
All
Core
core

Apache
Apache fork
Apache fork

- RLimitCPU
- RLimitNPROC
### RLimitNPROC

<table>
<thead>
<tr>
<th>number</th>
<th>max</th>
</tr>
</thead>
</table>

[, , , .htaccess]

- All
- Core
- core

---

**Apache**

- **Apache**
- **Apache fork**

**CGI ID**

- **RLimitMEM**
- **RLimitCPU**
Apache CGI
)

#!C:/Perl/bin/perl.exe

perl PATH:

#!perl

ScriptInterpreterSource Registry

Windows HKEY_CLASSES_ROOT
Shell\ExecCGI\Command Shell\Open\Command
Apache Script

ScriptInterpreterSource Registry ScriptAlias

Apache
Microsoft Internet Explorer

Apache 2.0 Registry-Strict Registry
Shell\ExecCGI\Command ExecCGI Windows
<table>
<thead>
<tr>
<th>Recommend</th>
<th>Determine if mod_status displays the first 63 characters of a request or the last 63, assuming the request itself is greater than 63 chars.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SeeRequestTail 0n</td>
</tr>
<tr>
<td></td>
<td>SeeRequestTail 0ff</td>
</tr>
<tr>
<td></td>
<td>Core</td>
</tr>
<tr>
<td></td>
<td>core</td>
</tr>
<tr>
<td></td>
<td>Available in Apache httpd 2.2.7 and later.</td>
</tr>
</tbody>
</table>
ServerAdmin

email-address

URL

Core

core

ServerAdmin

mailto:

ServerAdmin www-admin@foo.example.com
ServerAlias hostname [hostname] ...

Core

Apache
ServerName

simple.example.com DNS www.example.com

ServerName www.example.com:80

ServerName

IP

<VirtualHost>

ServerName

SSL

URL (mod_dir)

UseCanonicalPhysicalPort

- DNS_Apache
- Apache
- UseCanonicalName
- UseCanonicalPhysicalPort
- NameVirtualHost
- ServerAlias
ServerPath : URL
ServerPath URL-path

- Apache
ServerRoot directory-path
ServerRoot /usr/local/apache

Core

ServerRoot /home/httpd

- httpd -d
- ServerRoot
ServerSignature

ServerSignature Off

..., .htaccess

All

Core

core

ServerSignature (mod_info)

Off (Apache-1.2)

ServerName EMail ServerAdmin "mailto:"

2.0.44 ServerTokens

- ServerTokens
Server OS

**ServerTokens Prod[uctOnly]**

(): Server: Apache

**ServerTokens Major**

Server sends (e.g.): Server: Apache/2

**ServerTokens Minor**

Server sends (e.g.): Server: Apache/2.0

**ServerTokens Minimal**

(): Server: Apache/2.0.41

**ServerTokens OS**

(): Server: Apache/2.0.41 (Unix)

**ServerTokens Full ()**

(): Server: Apache/2.0.41 (Unix) PHP/4.2.2 MyMod/1.2

2.0.44 **ServerSignature**

- **ServerSignature**
SetHandler `handler-name` | None
.
, , .htaccess

FileInfo

Core
core

Apache 2.0 core

.htaccess

<Directory> <Location>

name

SetHandler imap-file

: URL http://servername/status

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

None SetHandler

: SetHandler (1) URL

- AddHandler
SetInputFilter

POST

SetInputFilter \textit{filter}[,\textit{filter}...]

\ldots\,.\,\texttt{.htaccess}

\ldots\,.\,\texttt{FileInfo}

\ldots\,.\,\texttt{Core}

\ldots\,.\,\texttt{core}

\texttt{SetInputFilter} \quad \textbf{POST}
SetOutputFilter

/www/data/  SSI

<Directory /www/data/>
  SetOutputFilter INCLUDES
</Directory>

SetOutputFilter

,...,.htaccess
FileInfo
Core
core
**TimeOut**: 60 seconds

**Core**

<table>
<thead>
<tr>
<th>I/O:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. TCP</td>
</tr>
<tr>
<td>2.</td>
</tr>
<tr>
<td>3. <strong>mod_cgi</strong> CGI</td>
</tr>
<tr>
<td>4. <strong>mod_ext_filter</strong></td>
</tr>
<tr>
<td>5. <strong>mod_proxy</strong></td>
</tr>
</tbody>
</table>
**Trace Enable**:

- TraceEnable <em>[on|off|extended]</em>
- TraceEnable on

---

**Core**

```
Core
```

---

**Apache**

```Apache
Apache 1.3.34, 2.0.55
```

---

**mod_proxy**

```
mod_proxy
```

---

**Trace Enable**

```
64k (Transfer-Encoding: chunked chunk) Apache chunk
```

---
Undefine the existence of a variable

UnDefine *parameter-name*

*Core*

*core*

- **Define**
- **IFDEF**
<table>
<thead>
<tr>
<th>UseCanonicalName</th>
<th>On</th>
<th>Off</th>
<th>Dns</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseCanonicalName</td>
<td>Off</td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td>,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>core</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Apache</th>
<th>URL</th>
<th>URL</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerName</td>
<td>( )</td>
<td>SERVER_NAME</td>
<td>SERVER_PORT</td>
</tr>
</tbody>
</table>

| UseCanonicalName | Off |
| CGI | SERVER_NAME | SERVER_PORT |

<table>
<thead>
<tr>
<th>www</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="http://www.domain.com/splat/">http://www.domain.com/splat/</a></td>
<td></td>
</tr>
</tbody>
</table>

| 1 | www.domain.com | 1 | -- | FAQ |

| UseCanonicalName | Off |
| CGI | SERVER_NAME | SERVER_PORT |

<table>
<thead>
<tr>
<th>Apache</th>
<th>IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>DNS</td>
</tr>
</tbody>
</table>

- UseCanonicalPhysicalPort
- ServerName
- Listen
<table>
<thead>
<tr>
<th>UseCanonicalPhysicalPort</th>
<th>On</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseCanonicalPhysicalPort</td>
<td>Off</td>
<td></td>
</tr>
<tr>
<td>Core</td>
<td></td>
<td></td>
</tr>
<tr>
<td>core</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>URL -- URL</th>
</tr>
</thead>
</table>
| UseCanonicalName 
| Off |

<table>
<thead>
<tr>
<th>UseCanonicalName On</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerName</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UseCanonicalName Off</th>
<th>DNS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Host:</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ServerName</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| UseCanonicalPhysicalPort Off |
|                             |
| UseCanonicalName            |
|                             |
- ServerName
- Listen
<VirtualHost>  </VirtualHost>
<VirtualHost>

- IP
- IP ()
- NameVirtualHost *
- IP
- IP IP

<VirtualHost 10.1.2.3>
    ServerAdmin webmaster@host.example.com
    DocumentRoot /www/docs/host.example.com
    ServerName host.example.com
    ErrorLog logs/host.example.com-error_log
    TransferLog logs/host.example.com-access_log
</VirtualHost>

IPv6

<VirtualHost [2001:db8::a00:20ff:fea7:ccea]>
    ServerAdmin webmaster@host.example.com
    DocumentRoot /www/docs/host.example.com
    ServerName host.example.com
    ErrorLog logs/host.example.com-error_log
    TransferLog logs/host.example.com-access_log
</VirtualHost>

IP
alias )
<VirtualHost> Apache Listen IP List
Apache listen

IP _default_
VirtualHost (_default_ )
:port
)

<VirtualHost> ServerName

- Apache
- DNS_Apache
- Apache

Copyright 2017 The Apache Software Foundation.
Apache License, Version 2.0
Apache MPM

(MPM)

MPM
CoreDumpDirectory directory

MPM
- beos, leader, mpm winnt, perchild, prefork, threadpool, worker

Apache

Linux
Apache root
2.4
EnableExceptionHook On|Off
EnableExceptionHook Off

MPM
leader, perchild, prefork, threadpool, worker
2.0.49

--enable-exception-hook configure

mod_whatchilledus n

Trawick EnableExceptionHook site
Graceful Shutdown Timeout

Graceful Shutdown Timeout: 0

MPM:
- prefork, worker, event
- 2.2

Graceful Shutdown Timeout: "graceful-stop"
Listen

Listen \[IP\text{-}address:]portnumber [protocol]

MPM
- beos, leader, mpm_netware, mpm_winnt, mpm_os2,
- perchild, prefork, threadpool, worker, event

Apache 2.0 protocol 2.1.5

Listen Apache IP listen

Listen

listen

80 8000

Listen 80
Listen 8000

IPv6

Listen [2001:db8::a00:20ff:fea7:cce0a]:80

protocol 443

AcceptFilter

protocol h1
Listen 192.170.2.1:8443 https

<table>
<thead>
<tr>
<th>IP</th>
<th>Listen</th>
</tr>
</thead>
</table>

- DNS
- Apache
<table>
<thead>
<tr>
<th>ListenBacklog backlog</th>
<th>ListenBacklog 511</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPM</td>
<td></td>
</tr>
<tr>
<td>beos, leader, mpm_netware, mpm_winnt, mpm_os2, perchild, prefork, threadpool, worker</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OS</th>
<th>OS</th>
<th>OS</th>
</tr>
</thead>
</table>
ListenCoresBucketsRatio ratio
ListenCoresBucketsRatio 0 (disabled)

MPM
event, worker, prefork

Available in Apache HTTP Server 2.4.17, with a kernel supporting the socket option SO_REUSEPORT and distributing new connections evenly across listening processes' (or threads') sockets using it (eg. Linux 3.9 and later, but not the current implementations of SO_REUSEPORT in *BSDs.
Limit on the number of connections that an individual child server will handle during its life

MaxConnectionsPerChild number

MaxConnectionsPerChild 0

MPM event, worker, prefork, mpm_winnt, mpm_netware, mpm_os2

Available Apache HTTP Server 2.3.9 and later. The old name MaxRequestsPerChild is still supported.
MaxMemFree: free()

MaxMemFree KBytes
MaxMemFree 0

MPM: beos, leader, mpm_netware, prefork, threadpool, worker, mpm_winnt

MaxMemFree free()
MaxRequestWorkers

Maximum number of connections that will be processed simultaneously

MaxRequestWorkers number

See usage for details

MPM

event, worker, prefork
MaxSpareThreads number

MPM
perchild MaxSpareThreads 10  MPM

worker, leader, threadpool MaxSpareThreads 250  MPM

mpm_netware MaxSpareThreads 100  M

beos  mpmt_os2  mpm_netware beos
MaxSpareThreads 50  mpmt_os2  10

MaxSpareThreads Apache

- perchild MaxSpareThreads ThreadLimit
- mpm_netware MinSpareThreads
- leader, threadpool, worker MinSpareThreads
  ThreadsPerChild
- MinSpareThreads
- StartServers
MinSpareThreads number

MPM

perchild

MinSpareThreads 5
NumServers 10 MinSpareThreads 5 50

worker, leader, threadpool

MinSpareThreads

mpm_netware

MinSpareThreads 10 MPM

beos  mpmt_os2  mpm_netware  beos
MinSpareThreads 1  mpmt_os2  5

- MaxSpareThreads
- StartServers
<table>
<thead>
<tr>
<th>ID</th>
<th>PidFile filename</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>logs/httpd.pid</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MPM</th>
</tr>
</thead>
<tbody>
<tr>
<td>beos, leader, mpm winnt, mpmt os2, perchild, prefork, threadpool, worker</td>
</tr>
</tbody>
</table>

**PidFile**  
/var/run/apache.pid

**ErrorLog**  
**TransferLog**

**PidFile**  
Apache 2  
apachectl ()
TCP

ReceiveBufferSize bytes

ReceiveBufferSize 0

MPM

beos, mpm_netware, mpm_winnt, mpm_os2, prefork, worker

TCP

0OS
ScoreBoardFile file-path

ScoreBoardFile logs/apache_status

MPM

beos, leader, mpm_winnt, perchild, prefork, threadpool, worker

Apache

Apache

ScoreBoardFile /var/run/apache_status

ScoreBoardFile RAM

• Apache
<table>
<thead>
<tr>
<th></th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SendBufferSize bytes</td>
</tr>
<tr>
<td></td>
<td>SendBufferSize 0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>MPM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>beos, leader, mpm_netware, mpm_winnt, mpm_os2, perchild, prefork, threadpool, worker</td>
</tr>
</tbody>
</table>

TCP

0OS
ServerLimit number

MPM

leader, perchild, prefork, threadpool, worker

prefork MPM

worker MPM

Apache

prefork MPM

MaxClients

MaxClients 256 ()

worker, leader, threadpool MPM

ThreadsPerChild

MaxClients

ThreadsPerChild 16 ()

perchild MPM

NumServers

NumServers 8 ()

ServerLimit 20000 (prefork)

Apache
StartServers number

MPM

leader, mpmt_os2, prefork, threadpool, worker

StartServers

MPM

leader, threadpool, worker

3 prefork 5 mpmt_os2 2
StartThreads number

MPM

beos, mpm_netware, perchild

perchild StartThreads 5

mpm_netware StartThreads 50

beos StartThreads 10
<table>
<thead>
<tr>
<th>MPM</th>
<th>Leader, mpm_winnt, perchild, threadpool, worker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apache 2.0.41</td>
<td>mpm_winnt</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Apache</th>
<th>ThreadsPerChild</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThreadLimit</td>
<td>Thr</td>
</tr>
<tr>
<td>ThreadsPerChild</td>
<td>Apach</td>
</tr>
<tr>
<td>ThreadLimit</td>
<td>mpm_winnt</td>
</tr>
<tr>
<td>1920</td>
<td>64</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ThreadLimit 20000</th>
<th>mpm_winnt</th>
</tr>
</thead>
<tbody>
<tr>
<td>ThreadLimit 1500</td>
<td></td>
</tr>
<tr>
<td>MPM</td>
<td>mpm_winnt</td>
</tr>
<tr>
<td>ThreadStackSize size</td>
<td></td>
</tr>
<tr>
<td>---------------------</td>
<td></td>
</tr>
<tr>
<td>NetWare 65536 OS</td>
<td></td>
</tr>
</tbody>
</table>

**MPM**
- leader
- mpm_netware
- mpm_winnt
- perchild
- threadpool
- worker

2.1

() (HP-UX)

ThreadStackSize OS

- Web
  - Web
  - ThreadStackSize
Apache MPM event

<table>
<thead>
<tr>
<th>Description</th>
<th>A variant of the worker MPM with the goal of consuming threads only for connections with active processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>MPM</td>
</tr>
<tr>
<td>Module Identifier</td>
<td>mpm_event_module</td>
</tr>
<tr>
<td>Source File</td>
<td>event.c</td>
</tr>
</tbody>
</table>

Summary

The event Multi-Processing Module (MPM) is designed to allow more requests to be served simultaneously by passing off some processing work to the listeners threads, freeing up the worker threads to serve new requests.

To use the event MPM, add --with-mpm=event to the configure script's arguments when building the httpd.

Bugfix checklist

- httpd changelog
- Known issues
- Report a bug

See also

- The worker MPM
**event** is based on the **worker** MPM, which implements a hybrid multi-process multi-threaded server. A single control process (the parent) is responsible for launching child processes. Each child process creates a fixed number of server threads as specified in the **ThreadsPerChild** directive, as well as a listener thread which listens for connections and passes them to a worker thread for processing when they arrive.

Run-time configuration directives are identical to those provided by **worker**, with the only addition of the **AsyncRequestWorkerFactor**.
This MPM tries to fix the 'keep alive problem' in HTTP. After a client completes the first request, it can keep the connection open, sending further requests using the same socket and saving significant overhead in creating TCP connections. However, Apache HTTP Server traditionally keeps an entire child process/thread waiting for data from the client, which brings its own disadvantages. To solve this problem, this MPM uses a dedicated listener thread for each process to handle both the Listening sockets, all sockets that are in a Keep Alive state, sockets where the handler and protocol filters have done their work and the ones where the only remaining thing to do is send the data to the client.

This new architecture, leveraging non-blocking sockets and modern kernel features exposed by APR (like Linux's epoll), no longer requires the mpm-accept Mutex configured to avoid the thundering herd problem.

The total amount of connections that a single process/threads block can handle is regulated by the AsyncRequestWorkerFactor directive.

Async connections

Async connections would need a fixed dedicated worker thread with the previous MPMs but not with event. The status page of mod_status shows new columns under the Async connections section:

Writing

While sending the response to the client, it might happen that the TCP write buffer fills up because the connection is too slow. Usually in this case a write() to the socket returns EWOULDBLOCK or EAGAIN, to become writable again after an
idle time. The worker holding the socket might be able to offload the waiting task to the listener thread, that in turn will re-assign it to the first idle worker thread available once an event will be raised for the socket (for example, "the socket is now writable"). Please check the Limitations section for more information.

**Keep-alive**

Keep Alive handling is the most basic improvement from the worker MPM. Once a worker thread finishes to flush the response to the client, it can offload the socket handling to the listener thread, that in turns will wait for any event from the OS, like "the socket is readable". If any new request comes from the client, then the listener will forward it to the first worker thread available. Conversely, if the **KeepAliveTimeout** occurs then the socket will be closed by the listener. In this way the worker threads are not responsible for idle sockets and they can be re-used to serve other requests.

**Closing**

Sometimes the MPM needs to perform a lingering close, namely sending back an early error to the client while it is still transmitting data to httpd. Sending the response and then closing the connection immediately is not the correct thing to do since the client (still trying to send the rest of the request) would get a connection reset and could not read the httpd's response. So in such cases, httpd tries to read the rest of the request to allow the client to consume the response. The lingering close is time bounded but it can take relatively long time, so a worker thread can offload this work to the listener.

These improvements are valid for both HTTP/HTTPS connections.

**Graceful process termination and Scoreboard usage**
This mpm showed some scalability bottlenecks in the past leading to the following error: "scoreboard is full, not at MaxRequestWorkers". MaxRequestWorkers limits the number of simultaneous requests that will be served at any given time and also the number of allowed processes (MaxRequestWorkers / ThreadsPerChild), meanwhile the Scoreboard is a representation of all the running processes and the status of their worker threads. If the scoreboard is full (so all the threads have a state that is not idle) but the number of active requests served is not MaxRequestWorkers, it means that some of them are blocking new requests that could be served but that are queued instead (up to the limit imposed by ListenBacklog). Most of the times the threads are stuck in the Graceful state, namely they are waiting to finish their work with a TCP connection to safely terminate and free up a scoreboard slot (for example handling long running requests, slow clients or connections with keep-alive enabled). Two scenarios are very common:

- During a graceful restart. The parent process signals all its children to complete their work and terminate, while it reloads the config and forks new processes. If the old children keep running for a while before stopping, the scoreboard will be partially occupied until their slots are freed.
- When the server load goes down in a way that causes httpd to stop some processes (for example due to MaxSpareThreads). This is particularly problematic because when the load increases again, httpd will try to start new processes. If the pattern repeats, the number of processes can rise quite a bit, ending up in a mixture of old processes trying to stop and new ones trying to do some work.

From 2.4.24 onward, mpm-event is smarter and it is able to handle graceful terminations in a much better way. Some of the improvements are:
- Allow the use of all the scoreboard slots up to `ServerLimit`. `MaxRequestWorkers` and `ThreadsPerChild` are used to limit the amount of active processes, meanwhile `ServerLimit` takes also into account the ones doing a graceful close to allow extra slots when needed. The idea is to use `ServerLimit` to instruct httpd about how many overall processes are tolerated before impacting the system resources.
- Force gracefully finishing processes to close their connections in keep-alive state.
- During graceful shutdown, if there are more running worker threads than open connections for a given process, terminate these threads to free resources faster (which may be needed for new processes).
- If the scoreboard is full, prevent more processes to finish gracefully due to reduced load until old processes have terminated (otherwise the situation would get worse once the load increases again).

The behavior described in the last point is completely observable via `mod_status` in the connection summary table through two new columns: "Slot" and "Stopping". The former indicates the PID and the latter if the process is stopping or not; the extra state "Yes (old gen)" indicates a process still running after a graceful restart.

**Limitations**

The improved connection handling may not work for certain connection filters that have declared themselves as incompatible with event. In these cases, this MPM will fall back to the behavior of the `worker` MPM and reserve one worker thread per connection. All modules shipped with the server are compatible with the event MPM.

A similar restriction is currently present for requests involving an
output filter that needs to read and/or modify the whole response body. If the connection to the client blocks while the filter is processing the data, and the amount of data produced by the filter is too big to be buffered in memory, the thread used for the request is not freed while httpd waits until the pending data is sent to the client.

To illustrate this point we can think about the following two situations: serving a static asset (like a CSS file) versus serving content retrieved from FCGI/CGI or a proxied server. The former is predictable, namely the event MPM has full visibility on the end of the content and it can use events: the worker thread serving the response content can flush the first bytes until EWOULDBLOCK or EAGAIN is returned, delegating the rest to the listener. This one in turn waits for an event on the socket, and delegates the work to flush the rest of the content to the first idle worker thread.

Meanwhile in the latter example (FCGI/CGI/proxied content) the MPM can't predict the end of the response and a worker thread has to finish its work before returning the control to the listener. The only alternative is to buffer the response in memory, but it wouldn't be the safest option for the sake of the server's stability and memory footprint.

**Background material**

The event model was made possible by the introduction of new APIs into the supported operating systems:

- epoll (Linux)
- kqueue (BSD)
- event ports (Solaris)

Before these new APIs where made available, the traditional select and poll APIs had to be used. Those APIs get slow if used to handle many connections or if the set of connections rate of change is high. The new APIs allow to monitor much more
connections and they perform way better when the set of connections to monitor changes frequently. So these APIs made it possible to write the event MPM, that scales much better with the typical HTTP pattern of many idle connections.

The MPM assumes that the underlying apr_pollset implementation is reasonably threadsafe. This enables the MPM to avoid excessive high level locking, or having to wake up the listener thread in order to send it a keep-alive socket. This is currently only compatible with KQueue and EPoll.
This MPM depends on APR's atomic compare-and-swap operations for thread synchronization. If you are compiling for an x86 target and you don't need to support 386s, or you are compiling for a SPARC and you don't need to run on pre-UltraSPARC chips, add --enable-nonportable-atomics=yes to the configure script's arguments. This will cause APR to implement atomic operations using efficient opcodes not available in older CPUs.

This MPM does not perform well on older platforms which lack good threading, but the requirement for EPoll or KQueue makes this moot.

- To use this MPM on FreeBSD, FreeBSD 5.3 or higher is recommended. However, it is possible to run this MPM on FreeBSD 5.2.1, if you use libkse (see man libmap.conf).
- For NetBSD, at least version 2.0 is recommended.
- For Linux, a 2.6 kernel is recommended. It is also necessary to ensure that your version of glibc has been compiled with support for EPoll.
<table>
<thead>
<tr>
<th>Description:</th>
<th>Limit concurrent connections per process</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>AsyncRequestWorkerFactor factor</td>
</tr>
<tr>
<td>Default:</td>
<td>2</td>
</tr>
<tr>
<td>Context:</td>
<td>server config</td>
</tr>
<tr>
<td>Status:</td>
<td>MPM</td>
</tr>
<tr>
<td>Module:</td>
<td>event</td>
</tr>
<tr>
<td>Compatibility:</td>
<td>Available in version 2.3.13 and later</td>
</tr>
</tbody>
</table>

The event MPM handles some connections in an asynchronous way, where request worker threads are only allocated for short periods of time as needed, and other connections with one request worker thread reserved per connection. This can lead to situations where all workers are tied up and no worker thread is available to handle new work on established async connections.

To mitigate this problem, the event MPM does two things:

- it limits the number of connections accepted per process, depending on the number of idle request workers;
- if all workers are busy, it will close connections in keep-alive state even if the keep-alive timeout has not expired. This allows the respective clients to reconnect to a different process which may still have worker threads available.

This directive can be used to fine-tune the per-process connection limit. A process will only accept new connections if the current number of connections (not counting connections in the "closing" state) is lower than:

\[
\text{ThreadsPerChild} + (\text{AsyncRequestWorkerFactor} \times \text{number of idle workers})
\]

An estimation of the maximum concurrent connections across all
the processes given an average value of idle worker threads can be calculated with:

\[
(\text{ThreadsPerChild} + (\text{AsyncRequestWorkerFactor} \times \text{number of idle workers})) \times \text{ServerLimit}
\]

Example

<table>
<thead>
<tr>
<th>ThreadsPerChild</th>
<th>ServerLimit</th>
<th>AsyncRequestWorkerFactor</th>
<th>MaxRequestWorkers</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>2</td>
<td>40</td>
</tr>
</tbody>
</table>

\[
\text{idle_workers} = 4 \text{ (average for all the processes to keep it simple)}
\]

\[
\text{max_connections} = (\text{ThreadsPerChild} + (\text{AsyncRequestWorkerFactor} \times \text{idle_workers})) \times \text{ServerLimit}
\]

\[
= (10 + (2 \times 4)) \times 4 = 72
\]

When all the worker threads are idle, then absolute maximum numbers of concurrent connections can be calculated in a simpler way:

\[
(\text{AsyncRequestWorkerFactor} + 1) \times \text{MaxRequestWorkers}
\]

Example

<table>
<thead>
<tr>
<th>ThreadsPerChild</th>
<th>ServerLimit</th>
<th>MaxRequestWorkers</th>
<th>AsyncRequestWorkerFactor</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>40</td>
<td>2</td>
</tr>
</tbody>
</table>

If all the processes have all threads idle then:

\[
\text{idle_workers} = 10
\]

We can calculate the absolute maximum numbers of concurrent connections in two ways:
Tuning AsyncRequestWorkerFactor requires knowledge about the traffic handled by httpd in each specific use case, so changing the default value requires extensive testing and data gathering from mod_status.

MaxRequestWorkers was called MaxClients prior to version 2.3.13. The above value shows that the old name did not accurately describe its meaning for the event MPM.

AsyncRequestWorkerFactor can take non-integer arguments, e.g "1.5".
**Apache MPM netware**

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

**Status:** MPM

**Module Identifier:** mpm_netware_module

**Source File:** mpm_netware.c

**Summary**

This Multi-Processing Module (MPM) implements an exclusively threaded web server that has been optimized for Novell NetWare.

The main thread is responsible for launching child worker threads which listen for connections and serve them when they arrive. Apache HTTP Server always tries to maintain several *spare* or idle worker threads, which stand ready to serve incoming requests. In this way, clients do not need to wait for a new child threads to be spawned before their requests can be served.

The **StartThreads**, **MinSpareThreads**, **MaxSpareThreads**, and **MaxThreads** regulate how the main thread creates worker threads to serve requests. In general, Apache httpd is very self-regulating, so most sites do not need to adjust these directives from their default values. Sites with limited memory may need to decrease **MaxThreads** to keep the server from thrashing (spawning and terminating idle threads). More information about tuning process creation is provided in the [performance hints](#) documentation.

**MaxConnectionsPerChild** controls how frequently the server recycles processes by killing old ones and launching new ones. On the NetWare OS it is highly recommended that this directive remain set to 0. This allows worker threads to continue servicing requests...
indefinitely.

**Bugfix checklist**
- httpd changelog
- Known issues
- Report a bug

**See also**
- Setting which addresses and ports Apache httpd uses
**Description:** Set the maximum number of worker threads

**Syntax:**    
MaxThreads *number*

**Default:**    
MaxThreads 2048

**Context:**    
server config

**Status:**    
MPM

**Module:**    
mpm_netware

The MaxThreads directive sets the desired maximum number worker threads allowable. The default value is also the compiled in hard limit. Therefore it can only be lowered, for example:

```
MaxThreads 512
```
Apache MPM os2

<table>
<thead>
<tr>
<th>Description:</th>
<th>Hybrid multi-process, multi-threaded MPM for OS/2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>MPM</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>mpm_mpmt_os2_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mpmt_os2.c</td>
</tr>
</tbody>
</table>

Summary

The Server consists of a main, parent process and a small, static number of child processes.

The parent process' job is to manage the child processes. This involves spawning children as required to ensure there are always StartServers processes accepting connections.

Each child process consists of a pool of worker threads and a main thread that accepts connections and passes them to the workers via a work queue. The worker thread pool is dynamic, managed by a maintenance thread so that the number of idle threads is kept between MinSpareThreads and MaxSpareThreads.

Bugfix checklist

httpd changelog
Known issues
Report a bug

See also
Setting which addresses and ports Apache uses
Apache MPM prefork

Bugfix checklist

httpd changelog
Known issues
Report a bug

Apache
listen

MaxClients Apache

() Unix 80 root

Apache

MaxRequestsPerChild
MaxSpareServers

MaxSpareServers 10

MPM

prefork

MaxSpareServers

kill

- MinSpareServers
- StartServers
MinSpareServers number
MinSpareServers 5

MPM
prefork

MaxSpareServers
1 1

- **MaxSpareServers**
- **StartServers**
Apache MPM winnt

Windows NT

MPM

mpm_winnt_module

mpm_winnt.c
Apache MPM worker

(MPM)

MPM
	ThreadsPerChild

mpm_worker_module

worker.c

Bugfix checklist

httpd changelog

Known issues

Report a bug

Apache
ThreadsPerChild

MinSpareThreads  MaxSpareThreads  fork

ThreadsPerChild

ThreadsPerChild  ThreadLimit

- MaxRequestsPerChild  0
- MaxSpareThreads  MaxClients

worker  MPM

ServerLimit 16
StartServers 2
MaxClients 150
MinSpareThreads 25
MaxSpareThreads 75
ThreadsPerChild 25

Unix  80  root

Apache

MaxRequestsPerChild
Apache mod_access_compat

| : ( IP ) |
| : Extension |
| : access_compat_module |
| : mod_access_compat.c |
| : Apache 2.3 (Apache 2.x ) |

mod_access_compat

.htaccess IP

Order Allow []

Note

mod_access_compat
mod_authz_default

(GET, PUT, POST)

Bugfix checklist
httpd changelog
Known issues
Report a bug

Require
mod_authz_host
mod_authz_core
Allow

from all

[host env=!]env-variable
[host env=!]env-variable

, .htaccess

Limit

Extension

mod_access_compat

Allow

from apache.org
Allow from .net example.edu

Apache HostnameLookups IP

IP

IP

IP

Allow from 10.1.2.3
Allow from 192.168.1.104 192.168.1.205

IP

IP
Allow from 10.1
Allow from 10 172.20 192.168.2

IP

Allow from 10.1.0.0/255.255.0.0
a.b.c.d w.x.y.z

/nnn CIDR

Allow from 10.1.0.0/16
nnn 1

IPv6 IPv6:

Allow from 2001:db8::a00:20ff:fea7:ccea
Allow from 2001:db8::a00:20ff:fea7:ccea/10

Allow variable
Allow from
variable
mod_setenvif
)

Referer HTTP

Example:
SetEnvIf User-Agent ^KnockKnock/2\0 let_me_in
<Directory /docroot>
  Order Deny,Allow
  Deny from all
  Allow from env=let_me_in
</Directory>

user-agent       KnockKnock/2.0
Deny from all|host|env=![env-variable]
[host|env=![env-variable] ... .htaccess
Limit
Extension
mod_access_compat
Order

Order allowing, .htaccess

Limit

Extension

mod_access_compat

Order  Allow Deny "3"

Deny)  2( Deny  Allow)

Allow  Deny

Order

Allow, Deny

Allow

Deny

Deny, Allow

Deny

Allow

Mutual-failure

Order Allow, Deny

<table>
<thead>
<tr>
<th></th>
<th>Allow,Deny</th>
<th>Deny,Allow</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allow</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deny</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2:</td>
<td>2:</td>
<td></td>
</tr>
<tr>
<td>Allow</td>
<td>Deny</td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
<td></td>
</tr>
</tbody>
</table>

**apache.org**

Order Deny, Allow
Deny from all
Allow from apache.org

**foo.apache.org**

Order Allow, Deny
Allow from apache.org
Deny from foo.apache.org

```
 Order Deny, Allow
 apache.org Deny from foo.apache.org
 apache.org Allow

Order

<Directory /www>
  Order Allow, Deny
</Directory>

 Order

 Deny /www
```

Order

```
 Directory .htaccess Allow Den
 Directory, Location, Files
```
Satisfy Any|All
Satisfy All
.htaccess
AuthConfig
Extension
mod_access_compat
2.0.51

<Limit>
<LimitExcept>

Allow Require
Any

Require valid-user
Allow from 192.168.1
Satisfy Any

2.0.51

<Limit> <LimitExcept>

- Allow
- Require
Apache mod_actions

Bugfix checklist

httpd changelog
Known issues
Report a bug

mod_cgi
CGI
Apache
Action action-type cgi-script [virtual]

.htaccess:
FileInfo
Base
mod_actions
virtual Apache 2.1

---

action-type                  cgi-script
AddHandler CGI URL-path
MIME                       URL
REDIRECT_HANDLER

# Requests for files of a particular MIME content type:
Action image/gif /cgi-bin/images.cgi

# Files of a particular file extension
AddHandler my-file-type .xyz
Action my-file-type /cgi-bin/program.cgi

MIME  image/gif          /cgi
2 .xyz                  /cgi-bin/pro

virtual

<Location /news>
  SetHandler news-handler
  Action news-handler /cgi-bin/news.cgi virtual
</Location>
- AddHandler
Script

```sh
method
cgi-script
```

AddHandler CGI URL-path
 PATH_INFO PATH_TRANSLATED

Script PUT

```sh
# For <ISINDEX>-style searching
Script GET /cgi-bin/search

# A CGI PUT handler
Script PUT /~bob/put.cgi
```
Apache mod_alias

- Base
- alias_module
- mod_alias.c

**Bugfix checklist**

- [httpd changelog](#)
- [Known issues](#)
- [Report a bug](#)

- mod_rewrite
Alias Redirect ( <VirtualHost> ) Alias Redirect

Alias Redirect Redirect RedirectMatch
Alias Alias Redirect

:

Alias /foo/bar /baz
Alias /foo /gaq

/foo Alias /foo/bar Alias
Alias
URL
Alias URL-path file-path|directory-path
Base
mod_alias

Alias /image /ftp/pub/image

http://myserver/image/foo.gif /ftp/pub/image

directory-filename

<Directory>
  (  <Location>

    Alias DocumentRoot

  </Location>

</Directory>

Alias /image /ftp/pub/image
<Directory /ftp/pub/image>
  Order allow,deny
  Allow from all
</Directory>
AliasMatch regex file-path|directory-path

Base

mod_alias

AliasMatch ^/icons(.*) /usr/local/apache/icons$1
Redirect URL

URL

Redirect [status] URL-path URL

, , , .htaccess

FileInfo

Base

mod_alias

Redirect URL

URL

Redirect /service http://foo2.bar.com/service

http://myserver/service/foo.txt
http://foo2.bar.com/service/foo.txt

Redirect Alias ScriptAlias .htaccess

<Directory> URL-path URL

status "temporary" (HTTP 302)

HTTP:

permanent (301)

temp (302)

seeother "See Other" (303)

gone "Gone" (410)
Status 300 399
(http_protocol.c send_error_response)

Redirect permanent /one http://example.com/two
Redirect 303 /three http://example.com/other
RedirectMatch (.*)\.gif$ http://www.anotherserver.com$1.jpg
RedirectPermanent URL-path URL

Redirect (301)
Redirect

RedirectTemp URL-path URL

FileInfo

Base

mod_alias

Redirect (302)
ScriptAlias URL-path file-path|directory-path

Base

mod_alias

ScriptAlias URL (%) URL-path

mod_cgi cgi-script

ScriptAlias /cgi-bin/ /web/cgi-bin/

http://myserver/cgi-bin/foo
ScriptAliasMatch `regex file-path|directory-path`

Base

mod_alias

ScriptAlias bin:

ScriptAliasMatch `^/cgi-bin(.*) /usr/local/apache/cgi-bin$1`
Apache Module mod_allowmethods

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Easily restrict what HTTP methods can be used on the server</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>allowmethods_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_allowmethods.c</td>
</tr>
</tbody>
</table>

**Summary**

This module makes it easy to restrict what HTTP methods can be used on a server. The most common configuration would be:

```xml
<Location "/">
  AllowMethods GET POST OPTIONS
</Location>
```
**Description:** Restrict access to the listed HTTP methods

**Syntax:**

```
AllowMethods reset | HTTP-method [HTTP-method] ...
```

**Default:**

```
AllowMethods reset
```

**Context:**

directory

**Status:**

Experimental

**Module:**

```
mod_allowmethods
```

The HTTP-methods are case sensitive and are generally, as per RFC, given in upper case. The GET and HEAD methods are treated as equivalent. The `reset` keyword can be used to turn off `mod_allowmethods` in a deeper nested context:

```
<Location "/svn">
  AllowMethods reset
</Location>
```

**Caution**

The TRACE method cannot be denied by this module; use `TraceEnable` instead.

`mod_allowmethods` was written to replace the rather kludgy implementation of `Limit` and `LimitExcept`. 

---

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](http://www.apache.org/licenses/LICENSE-2.0).
Apache mod_asis

send-as-is HTTP

Cgi nph

mime

httpd/send-as-is

Bugfix checklist

httpd changelog
Known issues
Report a bug

mod_headers
mod_cern_meta
Apache
send-as-is

AddHandler send-as-is asis

.asis Apache

CGI Status:

HTTP

as is ()

Status: 301 Now where did I leave that URL
Location: http://xyz.abc.com/foo/bar.html
Content-type: text/html

<html>
<head>
<title>Lame excuses'R'us</title>
</head>
<body>
<h1>Fred's exceptionally wonderful page has moved to Joe's site.</h1>
</body>
</html>
Bugfix checklist

httpd changelog
Known issues
Report a bug

AuthName
AuthType
Require
Reject
Satisfy (Deprecated)
<SatisfyAll>
<SatisfyOne>
Authentication howto
</SatisfyOne>
AuthBasicAuthoritative On|Off
AuthBasicAuthoritative On
.htaccess
AuthConfig
Base
mod_authbasic

AuthBasicProvider
AuthBasicAuthoritative Off
ID
AuthBasicProvider
Fake basic authentication using the given expressions for username and password

AuthBasicFake off|username [password]

none

,.htaccess

AuthConfig

Base

mod_auth_basic

Apache HTTP Server 2.4.5 and later
AuthBasicProvider

Example

```xml
<Location /secure>
  AuthType basic
  AuthName "private area"
  AuthBasicProvider dbm
  AuthDBMType SDBM
  AuthDBMUserFile /www/etc/dbmpasswd
  Require valid-user
</Location>
```

*mod_authn_dbm, mod_authn_file, mod_authn_dbd, mod_authnz_ldap*
Check passwords against the authentication providers as if Digest Authentication was in force instead of Basic Authentication.

- AuthBasicUseDigestAlgorithm MD5|Off
- AuthBasicUseDigestAlgorithm Off

.htaccess

AuthConfig
Base
mod_auth_basic
Apache HTTP Server 2.4.7 and later
Apache Module mod_auth_digest

**Description:** User authentication using MD5 Digest Authentication

**Status:** Extension

**Module Identifier:** auth_digest_module

**Source File:** mod_auth_digest.c

**Summary**

This module implements HTTP Digest Authentication ([RFC2617](https://tools.ietf.org/html/rfc2617)), and provides an alternative to `mod_auth_basic` where the password is not transmitted as cleartext. However, this does not lead to a significant security advantage over basic authentication. On the other hand, the password storage on the server is much less secure with digest authentication than with basic authentication. Therefore, using basic auth and encrypting the whole connection using `mod_ssl` is a much better alternative.

**Bugfix checklist**

- [httpd changelog](https://httpd.apache.org/changelog.html)
- [Known issues](https://httpd.apache.org/common/www/knownissues.html)
- [Report a bug](https://httpd.apache.org/reportbug.html)

**See also**

- [AuthName](https://httpd.apache.org/docs/currentmods/authname.html)
- [AuthType](https://httpd.apache.org/docs/currentmods/auth.html)
- [Require](https://httpd.apache.org/docs/currentmods/require.html)
Authentication howto
To use MD5 Digest authentication, configure the location to be protected as shown in the below example:

**Example:**

```
<Location "/private/">
  AuthType Digest
  AuthName "private area"
  AuthDigestDomain "/private/" "http://mirror.my.dom/private2,
  AuthDigestProvider file
  AuthUserFile "/web/auth/.digest_pw"
  Require valid-user
</Location>
```

**AuthDigestDomain** should list the locations that will be protected by this configuration.

The password file referenced in the **AuthUserFile** directive may be created and managed using the **htdigest** tool.

**Note**

Digest authentication was intended to be more secure than basic authentication, but no longer fulfills that design goal. A man-in-the-middle attacker can trivially force the browser to downgrade to basic authentication. And even a passive eavesdropper can brute-force the password using today's graphics hardware, because the hashing algorithm used by digest authentication is too fast. Another problem is that the storage of the passwords on the server is insecure. The contents of a stolen htdigest file can be used directly for digest authentication. Therefore using **mod_ssl** to encrypt the whole connection is strongly recommended.

**mod_auth_digest** only works properly on platforms where APR supports shared memory.
**Description:** Selects the algorithm used to calculate the challenge and response hashes in digest authentication.

**Syntax:** AuthDigestAlgorithm MD5|MD5-sess

**Default:** AuthDigestAlgorithm MD5

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_auth_digest

The `AuthDigestAlgorithm` directive selects the algorithm used to calculate the challenge and response hashes.

MD5-sess is not correctly implemented yet.
**Description:** Uris that are in the same protection space for digest authentication

**Syntax:**

```
AuthDigestDomain URI [URI] ...
```

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_auth_digest

The `AuthDigestDomain` directive allows you to specify one or more URIs which are in the same protection space (i.e. use the same realm and username/password info). The specified URIs are prefixes; the client will assume that all URIs "below" these are also protected by the same username/password. The URIs may be either absolute URIs (i.e. including a scheme, host, port, etc.) or relative URIs.

This directive *should* always be specified and contain at least the (set of) root URI(s) for this space. Omitting to do so will cause the client to send the Authorization header for *every request* sent to this server.

The URIs specified can also point to different servers, in which case clients (which understand this) will then share username/password info across multiple servers without prompting the user each time.
The **AuthDigestNonceLifetime** directive controls how long the server nonce is valid. When the client contacts the server using an expired nonce the server will send back a 401 with `stale=true`. If `seconds` is greater than 0 then it specifies the amount of time for which the nonce is valid; this should probably never be set to less than 10 seconds. If `seconds` is less than 0 then the nonce never expires.
**Description:** Sets the authentication provider(s) for this location

**Syntax:**
```
AuthDigestProvider provider-name
[provider-name] ...
```

**Default:** AuthDigestProvider file

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_auth_digest

The AuthDigestProvider directive sets which provider is used to authenticate the users for this location. The default file provider is implemented by the mod_authn_file module. Make sure that the chosen provider module is present in the server.

See [mod_authn_dbm](https://httpd.apache.org/docs/current/mod/mod_authn_dbm.html), [mod_authn_file](https://httpd.apache.org/docs/current/mod/mod_authn_file.html), [mod_authn_dbd](https://httpd.apache.org/docs/current/mod/mod_authn_dbd.html) and [mod_authn_socache](https://httpd.apache.org/docs/current/mod/mod_authn_socache.html) for providers.
<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>Determines the quality-of-protection to use in digest authentication</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
<td>AuthDigestQop none</td>
</tr>
<tr>
<td><strong>Default</strong></td>
<td>AuthDigestQop auth</td>
</tr>
<tr>
<td><strong>Context</strong></td>
<td>directory, .htaccess</td>
</tr>
<tr>
<td><strong>Override</strong></td>
<td>AuthConfig</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>mod_auth_digest</td>
</tr>
</tbody>
</table>

The `AuthDigestQop` directive determines the *quality-of-protection* to use. `auth` will only do authentication (username/password); `auth-int` is authentication plus integrity checking (an MD5 hash of the entity is also computed and checked); `none` will cause the module to use the old RFC-2069 digest algorithm (which does not include integrity checking). Both `auth` and `auth-int` may be specified, in which case the browser will choose which of these to use. `none` should only be used if the browser for some reason does not like the challenge it receives otherwise.

auth-int is not implemented yet.
**Description:** The amount of shared memory to allocate for keeping track of clients

**Syntax:** AuthDigestShmemSize size

**Default:** AuthDigestShmemSize 1000

**Context:** server config

**Status:** Extension

**Module:** mod_auth_digest

The AuthDigestShmemSize directive defines the amount of shared memory, that will be allocated at the server startup for keeping track of clients. Note that the shared memory segment cannot be set less than the space that is necessary for tracking at least one client. This value is dependent on your system. If you want to find out the exact value, you may simply set AuthDigestShmemSize to the value of 0 and read the error message after trying to start the server.

The size is normally expressed in Bytes, but you may follow the number with a K or an M to express your value as KBytes or MBytes. For example, the following directives are all equivalent:

- AuthDigestShmemSize 1048576
- AuthDigestShmemSize 1024K
- AuthDigestShmemSize 1M
Apache Module mod_auth_form

**Description:** Form authentication

**Status:** Base

**Module Identifier:** auth_form_module

**Source File:** mod_auth_form.c

**Compatibility:** Available in Apache 2.3 and later

---

**Summary**

**Warning**

Form authentication depends on the `mod_session` modules, and these modules make use of HTTP cookies, and as such can fall victim to Cross Site Scripting attacks, or expose potentially private information to clients. Please ensure that the relevant risks have been taken into account before enabling the session functionality on your server.

This module allows the use of an HTML login form to restrict access by looking up users in the given providers. HTML forms require significantly more configuration than the alternatives, however an HTML login form can provide a much friendlier experience for end users.

HTTP basic authentication is provided by `mod_auth_basic`, and HTTP digest authentication is provided by `mod_auth_digest`. This module should be combined with at least one authentication module such as `mod_authn_file` and one authorization module such as `mod_authz_user`.

Once the user has been successfully authenticated, the user's login details will be stored in a session provided by `mod_session`. 
Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod_session
AuthName
AuthType
Require
Authentication howto
To protect a particular URL with `mod_auth_form`, you need to decide where you will store your `session`, and you will need to decide what method you will use to authenticate. In this simple example, the login details will be stored in a session based on `mod_session_cookie`, and authentication will be attempted against a file using `mod_authn_file`. If authentication is unsuccessful, the user will be redirected to the form login page.

**Basic example**

```html
<Location "/admin">
    AuthFormProvider file
    AuthUserFile "conf/passwd"
    AuthType form
    AuthName "/admin"
    AuthFormLoginRequiredLocation "http://example.com/login.html"

    Session On
    SessionCookieName session path=/

    Require valid-user
</Location>
```

The directive `AuthType` will enable the `mod_auth_form` authentication when set to the value `form`. The directives `AuthFormProvider` and `AuthUserFile` specify that usernames and passwords should be checked against the chosen file.

The directives `Session` and `SessionCookieName` store the session within an HTTP cookie on the browser. For more information on the different options for configuring a session, read the documentation for `mod_session`.

You can optionally add a `SessionCryptoPassphrase` to create an encrypted session cookie. This required the additional module
**mod_session_crypto** be loaded.

In the simple example above, a URL has been protected by **mod_auth_form**, but the user has yet to be given an opportunity to enter their username and password. Options for doing so include providing a dedicated standalone login page for this purpose, or for providing the login page inline.
The login form can be hosted as a standalone page, or can be provided inline on the same page.

When configuring the login as a standalone page, unsuccessful authentication attempts should be redirected to a login form created by the website for this purpose, using the `AuthFormLoginRequiredLocation` directive. Typically this login page will contain an HTML form, asking the user to provide their username and password.

### Example login form

```html
<form method="POST" action="/dologin.html">
  Username: <input type="text" name="httpd_username" value="" />
  Password: <input type="password" name="httpd_password" value= "" />
  <input type="submit" name="login" value="Login" />
</form>
```

The part that does the actual login is handled by the `form-login-handler`. The action of the form should point at this handler, which is configured within Apache httpd as follows:

### Form login handler example

```http
c<Location "/dologin.html">
  SetHandler form-login-handler
  AuthFormLoginRequiredLocation "http://example.com/login.html"
  AuthFormLoginSuccessLocation "http://example.com/admin/index.html"
  AuthFormProvider file
  AuthUserFile "conf/passwd"
  AuthType form
  AuthName /admin
  Session On
  SessionCookieName session path=/
</Location>
```

The URLs specified by the `AuthFormLoginRequiredLocation` directive will typically point to a page explaining to the user that
their login attempt was unsuccessful, and they should try again. The `AuthFormLoginSuccessLocation` directive specifies the URL the user should be redirected to upon successful login.

Alternatively, the URL to redirect the user to on success can be embedded within the login form, as in the example below. As a result, the same `form-login-handler` can be reused for different areas of a website.

**Example login form with location**

```html
<form method="POST" action="/dologin.html">
    Username: <input type="text" name="httpd_username" value="" />
    Password: <input type="password" name="httpd_password" value="" />
    <input type="submit" name="login" value="Login" />
    <input type="hidden" name="httpd_location" value="http://example.com/success.html" />
</form>
```
Warning

A risk exists that under certain circumstances, the login form configured using inline login may be submitted more than once, revealing login credentials to the application running underneath. The administrator must ensure that the underlying application is properly secured to prevent abuse. If in doubt, use the standalone login configuration.

As an alternative to having a dedicated login page for a website, it is possible to configure `mod_auth_form` to authenticate users inline, without being redirected to another page. This allows the state of the current page to be preserved during the login attempt. This can be useful in a situation where a time limited session is in force, and the session times out in the middle of the user request. The user can be re-authenticated in place, and they can continue where they left off.

If a non-authenticated user attempts to access a page protected by `mod_auth_form` that isn't configured with a `AuthFormLoginRequiredLocation` directive, a `HTTP_UNAUTHORIZED` status code is returned to the browser indicating to the user that they are not authorized to view the page.

To configure inline authentication, the administrator overrides the error document returned by the `HTTP_UNAUTHORIZED` status code with a custom error document containing the login form, as follows:

**Basic inline example**

```plaintext
AuthFormProvider file
ErrorDocument 401 "/login.shtml"
AuthUserFile "conf/passwd"
AuthType form
```
The error document page should contain a login form with an empty action property, as per the example below. This has the effect of submitting the form to the original protected URL, without the page having to know what that URL is.

**Example inline login form**

```html
<form method="POST" action="">
  Username: <input type="text" name="httpd_username" value="" />
  Password: <input type="password" name="httpd_password" value="" />
  <input type="submit" name="login" value="Login" />
</form>
```

When the end user has filled in their login details, the form will make an HTTP POST request to the original password protected URL. **mod_auth_form** will intercept this POST request, and if HTML fields are found present for the username and password, the user will be logged in, and the original password protected URL will be returned to the user as a GET request.
A limitation of the inline login technique described above is that should an HTML form POST have resulted in the request to authenticate or reauthenticate, the contents of the original form posted by the browser will be lost. Depending on the function of the website, this could present significant inconvenience for the end user.

`mod_auth_form` addresses this by allowing the method and body of the original request to be embedded in the login form. If authentication is successful, the original method and body will be retried by Apache httpd, preserving the state of the original request.

To enable body preservation, add three additional fields to the login form as per the example below.

**Example with body preservation**

```
<form method="POST" action="">
    Username: <input type="text" name="httpd_username" value="" />
    Password: <input type="password" name="httpd_password" value="" />
    <input type="submit" name="login" value="Login" />
    <input type="hidden" name="httpd_method" value="POST" />
    <input type="hidden" name="httpd_mimetype" value="application/x-www-form-urlencoded" />
    <input type="hidden" name="httpd_body" value="name1=value1&name2=value2" />
</form>
```

How the method, mimetype and body of the original request are embedded within the login form will depend on the platform and technology being used within the website.

One option is to use the `mod_include` module along with the `KeptBodySize` directive, along with a suitable CGI script to embed the variables in the form.
Another option is to render the login form using a CGI script or other dynamic technology.

**CGI example**

```bash
AuthFormProvider file
ErrorDocument 401 "/cgi-bin/login.cgi"
...
To enable a user to log out of a particular session, configure a page to be handled by the `form-logout-handler`. Any attempt to access this URL will cause the username and password to be removed from the current session, effectively logging the user out.

By setting the `AuthFormLogoutLocation` directive, a URL can be specified that the browser will be redirected to on successful logout. This URL might explain to the user that they have been logged out, and give the user the option to log in again.

**Basic logout example**

```
SetHandler form-logout-handler
AuthName realm
AuthFormLogoutLocation "http://example.com/loggedout.html"
Session On
SessionCookieName session path=/
```

Note that logging a user out does not delete the session; it merely removes the username and password from the session. If this results in an empty session, the net effect will be the removal of that session, but this is not guaranteed. If you want to guarantee the removal of a session, set the `SessionMaxAge` directive to a small value, like 1 (setting the directive to zero would mean no session age limit).

**Basic session expiry example**

```
SetHandler form-logout-handler
AuthFormLogoutLocation "http://example.com/loggedout.html"
Session On
SessionMaxAge 1
SessionCookieName session path=/
```
Note that form submission involves URLEncoding the form data: in this case the username and password. You should therefore pick usernames and passwords that avoid characters that are URLencoded in form submission, or you may get unexpected results.
**Description:** Sets whether authorization and authentication are passed to lower level modules

**Syntax:** AuthFormAuthoritative On|Off

**Default:** AuthFormAuthoritative On

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_auth_form

Normally, each authorization module listed in **AuthFormProvider** will attempt to verify the user, and if the user is not found in any provider, access will be denied. Setting the **AuthFormAuthoritative** directive explicitly to Off allows for both authentication and authorization to be passed on to other non-provider-based modules if there is no **userID** or **rule** matching the supplied userID. This should only be necessary when combining **mod_auth_form** with third-party modules that are not configured with the **AuthFormProvider** directive. When using such modules, the order of processing is determined in the modules' source code and is not configurable.
**Description:** The name of a form field carrying the body of the request to attempt on successful login

**Syntax:** AuthFormBody fieldname

**Default:** httpd_body

**Context:** directory

**Status:** Base

**Module:** mod_auth_form

**Compatibility:** Available in Apache HTTP Server 2.3.0 and later

The **AuthFormMethod** directive specifies the name of an HTML field which, if present, will contain the method of the request to submit should login be successful.

By populating the form with fields described by **AuthFormMethod**, **AuthFormMimetype** and **AuthFormBody**, a website can retry a request that may have been interrupted by the login screen, or by a session timeout.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Disable the CacheControl no-store header on the login page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>AuthFormDisableNoStore  *On</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>AuthFormDisableNoStore <em>Off</em></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>directory</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_auth_form</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache HTTP Server 2.3.0 and later</td>
</tr>
</tbody>
</table>

The `AuthFormDisableNoStore` flag disables the sending of a Cache-Control `no-store` header with the error 401 page returned when the user is not yet logged in. The purpose of the header is to make it difficult for an ecmascript application to attempt to resubmit the login form, and reveal the username and password to the backend application. Disable at your own risk.
The **AuthFormFakeBasicAuth** flag determines whether a Basic Authentication header will be added to the request headers. This can be used to expose the username and password to an underlying application, without the underlying application having to be aware of how the login was achieved.
**Description:** The name of a form field carrying a URL to redirect to on successful login

**Syntax:** AuthFormLocation *fieldname*

**Default:** httpd_location

**Context:** directory

**Status:** Base

**Module:** mod_auth_form

**Compatibility:** Available in Apache HTTP Server 2.3.0 and later

The **AuthFormLocation** directive specifies the name of an HTML field which, if present, will contain a URL to redirect the browser to should login be successful.
The **AuthFormLoginRequiredLocation** directive specifies the URL to redirect to should the user not be authorised to view a page. The value is parsed using the `ap_expr` parser before being sent to the client. By default, if a user is not authorised to view a page, the HTTP response code `HTTP_UNAUTHORIZED` will be returned with the page specified by the **ErrorDocument** directive. This directive overrides this default.

Use this directive if you have a dedicated login page to redirect users to.
The `AuthFormLoginSuccessLocation` directive specifies the URL to redirect to should the user have logged in successfully. The value is parsed using the `ap_expr` parser before being sent to the client. This directive can be overridden if a form field has been defined containing another URL using the `AuthFormLocation` directive.

Use this directive if you have a dedicated login URL, and you have not embedded the destination page in the login form.
**Description:** The URL to redirect to after a user has logged out

**Syntax:** AuthFormLogoutLocation *uri*

**Default:** none

**Context:** directory

**Status:** Base

**Module:** mod_auth_form

**Compatibility:** Available in Apache HTTP Server 2.3.0 and later. The use of the expression parser has been added in 2.4.4.

The `AuthFormLogoutLocation` directive specifies the URL of a page on the server to redirect to should the user attempt to log out. The value is parsed using the `ap_expr` parser before being sent to the client.

When a URI is accessed that is served by the handler `form-logout-handler`, the page specified by this directive will be shown to the end user. For example:

```
<Location "/logout">
  SetHandler form-logout-handler
  AuthFormLogoutLocation "http://example.com/loggedout.html"
  Session on
  #...
</Location>
```

An attempt to access the URI `/logout/` will result in the user being logged out, and the page `/loggedout.html` will be displayed. Make sure that the page `loggedout.html` is not password protected, otherwise the page will not be displayed.
**Description:** The name of a form field carrying the method of the request to attempt on successful login

**Syntax:** `AuthFormMethod fieldname`

**Default:** `httpd_method`

**Context:** `directory`

**Status:** `Base`

**Module:** `mod_auth_form`

**Compatibility:** Available in Apache HTTP Server 2.3.0 and later

The **AuthFormMethod** directive specifies the name of an HTML field which, if present, will contain the method of the request to submit should login be successful.

By populating the form with fields described by **AuthFormMethod**, **AuthFormMimetype** and **AuthFormBody**, a website can retry a request that may have been interrupted by the login screen, or by a session timeout.
**Description:** The name of a form field carrying the mimetype of the body of the request to attempt on successful login

**Syntax:**

`AuthFormMimetype fieldname`

**Default:**

`httpd_mimetype`

**Context:**

directory

**Status:**

Base

**Module:**

`mod_auth_form`

**Compatibility:** Available in Apache HTTP Server 2.3.0 and later

The **AuthFormMethod** directive specifies the name of an HTML field which, if present, will contain the mimetype of the request to submit should login be successful.

By populating the form with fields described by **AuthFormMethod**, **AuthFormMimetype** and **AuthFormBody**, a website can retry a request that may have been interrupted by the login screen, or by a session timeout.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>The name of a form field carrying the login password</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>AuthFormPassword <em>fieldname</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>httpd_password</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>directory</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_auth_form</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache HTTP Server 2.3.0 and later</td>
</tr>
</tbody>
</table>

The **AuthFormPassword** directive specifies the name of an HTML field which, if present, will contain the password to be used to log in.
**Description:** Sets the authentication provider(s) for this location

**Syntax:**
```
AuthFormProvider provider-name
[provider-name] ...
```

**Default:** AuthFormProvider file

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_auth_form

The `AuthFormProvider` directive sets which provider is used to authenticate the users for this location. The default file provider is implemented by the `mod_authn_file` module. Make sure that the chosen provider module is present in the server.

**Example**

```html
<Location "/secure">
    AuthType form
    AuthName "private area"
    AuthFormProvider dbm
    AuthDBMType SDBM
    AuthDBMUserFile "/www/etc/dbmpasswd"
    Require valid-user
    #...
</Location>
```

Providers are implemented by `mod_authn_dbm`, `mod_authn_file`, `mod_authn_dbd`, `mod_authnz_ldap` and `mod_authn_socache`. 
**Description:** Bypass authentication checks for high traffic sites

**Syntax:** `AuthFormSitePassphrase secret`

**Default:** none

**Context:** directory

**Status:** Base

**Module:** `mod_auth_form`

**Compatibility:** Available in Apache HTTP Server 2.3.0 and later

The `AuthFormSitePassphrase` directive specifies a passphrase which, if present in the user session, causes Apache httpd to bypass authentication checks for the given URL. It can be used on high traffic websites to reduce the load induced on authentication infrastructure.

The passphrase can be inserted into a user session by adding this directive to the configuration for the `form-login-handler`. The `form-login-handler` itself will always run the authentication checks, regardless of whether a passphrase is specified or not.

**Warning**

If the session is exposed to the user through the use of `mod_session_cookie`, and the session is not protected with `mod_session_crypto`, the passphrase is open to potential exposure through a dictionary attack. Regardless of how the session is configured, ensure that this directive is not used within URL spaces where private user data could be exposed, or sensitive transactions can be conducted. Use at own risk.
The `AuthFormSize` directive specifies the maximum size of the body of the request that will be parsed to find the login form.

If a login request arrives that exceeds this size, the whole request will be aborted with the HTTP response code `HTTP_REQUEST_TOO_LARGE`.

If you have populated the form with fields described by `AuthFormMethod`, `AuthFormMimetype` and `AuthFormBody`, you probably want to set this field to a similar size as the `KeptBodySize` directive.
**Description:** The name of a form field carrying the login username

**Syntax:** AuthFormUsername *fieldname*

**Default:** httpd_username

**Context:** directory

**Status:** Base

**Module:** mod_auth_form

**Compatibility:** Available in Apache HTTP Server 2.3.0 and later

The **AuthFormUsername** directive specifies the name of an HTML field which, if present, will contain the username to be used to log in.
Apache mod_authn_anon

- "anonymous"
- Extension
- authn_anon_module
- mod_authn_anon.c
- Apache 2.1

mod_auth_basic anonymous-ftp ID

'anonymous'

mod_auth_basic AuthBasicProvider
htpasswd

- userID
- (Anonymous_NoUserID)
- (Anonymous_MustGiveEmail) '@'
- userID
- anonymous guest www test welcome

```
<Directory /var/www/html/private>
  AuthName "Use 'anonymous' & Email address for guest entry"
  AuthType Basic
  AuthBasicProvider file anon
  AuthUserFile /path/to/your/.htpasswd

  Anonymous_NoUserID off
  Anonymous_MustGiveEmail on
  Anonymous_VerifyEmail on
  Anonymous_LogEmail on
  Anonymous anonymous guest www test welcome

  Require valid-user
</Directory>
```
| userID | Anonymous user [user] ...
| .htaccess | AuthConfig Extension
| mod_authn_anon |

| userID | 'anonymous' userID |

| | Anonymous anonymous "Not Registered" "I don't know"

| userID "anonymous", "AnonymMous", "Not Registered", "I Don't Know" |

<p>| Apache 2.1 userID &quot;*&quot; |</p>
<table>
<thead>
<tr>
<th>Anonymous LogEmail On/Off</th>
<th>Anonymous LogEmail On</th>
<th>.htaccess</th>
<th>AuthConfig</th>
<th>Extension</th>
<th>mod_authn_anon</th>
</tr>
</thead>
<tbody>
<tr>
<td>On</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


Anonymous_MustGiveEmail On|Off
Anonymous_MustGiveEmail On
.htaccess
AuthConfig
Extension
mod_authn_anon
<table>
<thead>
<tr>
<th>userID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous_NoUserID On</td>
</tr>
<tr>
<td>Anonymous_NoUserID                      Off</td>
</tr>
<tr>
<td>.htaccess</td>
</tr>
<tr>
<td>AuthConfig</td>
</tr>
<tr>
<td>Extension</td>
</tr>
<tr>
<td>mod_authn_anon</td>
</tr>
</tbody>
</table>

On userID ( )
Anonymous_VerifyEmail On|Off
Anonymous_VerifyEmail Off
.htaccess
AuthConfig
Extension
mod_authn_anon

On

)
Apache Module mod_authn_core

| Description: | Core Authentication          |
| Status:      | Base                         |
| Module Identifier: | authn_core_module           |
| Source File: | mod_authn_core.c             |
| Compatibility: | Available in Apache 2.3 and later |

Summary

This module provides core authentication capabilities to allow or deny access to portions of the web site. mod_authn_core provides directives that are common to all authentication providers.
Extended authentication providers can be created within the configuration file and assigned an alias name. The alias providers can then be referenced through the directives `AuthBasicProvider` or `AuthDigestProvider` in the same way as a base authentication provider. Besides the ability to create and alias an extended provider, it also allows the same extended authentication provider to be reference by multiple locations.

**Examples**

This example checks for passwords in two different text files.

**Checking multiple text password files**

```
# Check here first
<AuthnProviderAlias file file1>
    AuthUserFile "/www/conf/passwords1"
</AuthnProviderAlias>

# Then check here
<AuthnProviderAlias file file2>
    AuthUserFile "/www/conf/passwords2"
</AuthnProviderAlias>

<Directory "/var/web/pages/secure">
    AuthBasicProvider file1 file2
    AuthType Basic
    AuthName "Protected Area"
    Require valid-user
</Directory>
```

The example below creates two different ldap authentication provider aliases based on the ldap provider. This allows a single authenticated location to be serviced by multiple ldap hosts:

**Checking multiple LDAP servers**

```
<AuthnProviderAlias ldap ldap-alias1>
    AuthLDAPBindDN cn=youruser,o=ctx
    AuthLDAPBindPassword yourpassword
```
AuthLDAPURL ldap://ldap.host/o=ctx
</AuthnProviderAlias>
<AuthnProviderAlias ldap ldap-other-alias>
  AuthLDAPBindDN cn=yourotheruser,o=dev
  AuthLDAPBindPassword yourotherpassword
  AuthLDAPURL ldap://other.ldap.host/o=dev?cn
</AuthnProviderAlias>

Alias "/secure" "/webpages/secure"
<Directory "/webpages/secure">
  AuthBasicProvider ldap-other-alias ldap-alias1
  AuthType Basic
  AuthName "LDAP Protected Place"
  Require valid-user
  # Note that Require ldap-* would not work here, since the
  # AuthnProviderAlias does not provide the config to authori:
  # that are implemented in the same module as the authentica
**Description:** Authorization realm for use in HTTP authentication

**Syntax:** `AuthName auth-domain`

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_authn_core

This directive sets the name of the authorization realm for a directory. This realm is given to the client so that the user knows which username and password to send. `AuthName` takes a single argument; if the realm name contains spaces, it must be enclosed in quotation marks. It must be accompanied by `AuthType` and `Require` directives, and directives such as `AuthUserFile` and `AuthGroupFile` to work.

For example:

```
AuthName "Top Secret"
```

The string provided for the `AuthName` is what will appear in the password dialog provided by most browsers.

**See also**

- [Authentication, Authorization, and Access Control](#)
- [mod_authz_core](#)
**Description:** Enclose a group of directives that represent an extension of a base authentication provider and referenced by the specified alias.

**Syntax:** `<AuthnProviderAlias baseProvider Alias> ... </AuthnProviderAlias>`

**Context:** server config

**Status:** Base

**Module:** mod_authn_core

`<AuthnProviderAlias>` and `</AuthnProviderAlias>` are used to enclose a group of authentication directives that can be referenced by the alias name using one of the directives `AuthBasicProvider` or `AuthDigestProvider`.

This directive has no affect on authorization, even for modules that provide both authentication and authorization.
This directive selects the type of user authentication for a directory. The authentication types available are None, Basic (implemented by `mod_authn_file`), Digest (implemented by `mod_authz_user`), and Form (implemented by `mod_authn_form`).

To implement authentication, you must also use the `AuthName` and `Require` directives. In addition, the server must have an authentication-provider module such as `mod_authn_file` and an authorization module such as `mod_authz_user`.

The authentication type None disables authentication. When authentication is enabled, it is normally inherited by each subsequent configuration section, unless a different authentication type is specified. If no authentication is desired for a subsection of an authenticated section, the authentication type None may be used; in the following example, clients may access the `/www/docs/public` directory without authenticating:

```apache
<Directory "/www/docs">
  AuthType Basic
  AuthName Documents
  AuthBasicProvider file
  AuthUserFile "/usr/local/apache/passwd/
  Require valid-user
</Directory>
```
When disabling authentication, note that clients which have already authenticated against another portion of the server's document tree will typically continue to send authentication HTTP headers or cookies with each request, regardless of whether the server actually requires authentication for every resource.

See also

- Authentication, Authorization, and Access Control
Apache Module mod_authn_dbd

**Description:** User authentication using an SQL database

**Status:** Extension

**Module Identifier:** authn_dbd_module

**Source File:** mod_authn_dbd.c

**Compatibility:** Available in Apache 2.1 and later

**Summary**

This module provides authentication front-ends such as `mod_auth_digest` and `mod_auth_basic` to authenticate users by looking up users in SQL tables. Similar functionality is provided by, for example, `mod_authn_file`.

This module relies on `mod_dbd` to specify the backend database driver and connection parameters, and manage the database connections.

When using `mod_auth_basic` or `mod_auth_digest`, this module is invoked via the `AuthBasicProvider` or `AuthDigestProvider` with the dbd value.

**Bugfix checklist**

- [httpd changelog](#)
- [Known issues](#)
- [Report a bug](#)
See also

AuthName
AuthType
AuthBasicProvider
AuthDigestProvider
DBDriver
DBDParams
Password Formats
Some users of DBD authentication in HTTPD 2.2/2.4 have reported that it imposes a problematic load on the database. This is most likely where an HTML page contains hundreds of objects (e.g. images, scripts, etc) each of which requires authentication. Users affected (or concerned) by this kind of problem should use `mod_authn_socache` to cache credentials and take most of the load off the database.
This simple example shows use of this module in the context of the Authentication and DBD frameworks.

```plaintext
# mod_dbd configuration
# UPDATED to include authentication caching
DBDriver pgsql
DBDParams "dbname=apacheauth user=apache password=xxxxxx"
DBDMin 4
DBDKeep 8
DBDMax 20
DBDExptime 300

<Directory "/usr/www/myhost/private">
    # mod_authn_core and mod_auth_basic configuration
    # for mod_authn_dbd
    AuthType Basic
    AuthName "My Server"
    # To cache credentials, put socache ahead
    AuthBasicProvider socache ddbd
    # Also required for caching: tell the cache how to handle dbd
    AuthnCacheProvideFor ddbd
    AuthnCacheContext my-server
    # mod_authz_core configuration
    Require valid-user
    # mod_authn_dbd SQL query to authenticate
    AuthDBDUserPWQuery "SELECT password FROM authn WHERE user = \%s"
</Directory>
```
If httpd was built against APR version 1.3.0 or higher, then whenever a query is made to the database server, all column values in the first row returned by the query are placed in the environment, using environment variables with the prefix "AUTHENTICATE_".

If a database query for example returned the username, full name and telephone number of a user, a CGI program will have access to this information without the need to make a second independent database query to gather this additional information.

This has the potential to dramatically simplify the coding and configuration required in some web applications.
**Description:** SQL query to look up a password for a user

**Syntax:** `AuthDBDUserPWQuery query`

**Context:** directory

**Status:** Extension

**Module:** `mod_authn_dbd`

The **AuthDBDUserPWQuery** specifies an SQL query to look up a password for a specified user. The user's ID will be passed as a single string parameter when the SQL query is executed. It may be referenced within the query statement using a `%s` format specifier.

```
AuthDBDUserPWQuery "SELECT password FROM authn WHERE user = %s"
```

The first column value of the first row returned by the query statement should be a string containing the encrypted password. Subsequent rows will be ignored. If no rows are returned, the user will not be authenticated through `mod_authn_dbd`.

If `httpd` was built against [APR](https://apr.apache.org) version 1.3.0 or higher, any additional column values in the first row returned by the query statement will be stored as environment variables with names of the form `AUTHENTICATE_COLUMN`.

The encrypted password format depends on which authentication frontend (e.g. `mod_auth_basic` or `mod_auth_digest`) is being used. See [Password Formats](#) for more information.
**Description:** SQL query to look up a password hash for a user and realm.

**Syntax:** AuthDBDUserRealmQuery query

**Context:** directory

**Status:** Extension

**Module:** mod_authn_dbd

The AuthDBDUserRealmQuery specifies an SQL query to look up a password for a specified user and realm in a digest authentication process. The user's ID and the realm, in that order, will be passed as string parameters when the SQL query is executed. They may be referenced within the query statement using %s format specifiers.

```
AuthDBDUserRealmQuery "SELECT password FROM authn WHERE user = %s AND realm = %s"
```

The first column value of the first row returned by the query statement should be a string containing the encrypted password. Subsequent rows will be ignored. If no rows are returned, the user will not be authenticated through mod_authn_dbd.

If httpd was built against APR version 1.3.0 or higher, any additional column values in the first row returned by the query statement will be stored as environment variables with names of the form AUTHENTICATE_COLUMN.

The encrypted password format depends on which authentication frontend (e.g. mod_auth_basic or mod_auth_digest) is being used. See Password Formats for more information.
Apache mod_authn_dbm

DBM
Extension
authn_dbm_module
mod_authn_dbm.c
Apache 2.1

mod_auth_digest  mod_auth_basic
mod_authn_file

mod_auth_basic  mod_auth_digest
AuthDigestProvider  dbm

Bugfix checklist
httpd changelog
Known issues
Report a bug

AuthName
AuthType
AuthBasicProvider
AuthDigestProvider
AuthDBMType default|SDBM|GDBM|NDBM|DB

AuthDBMType default

.htaccess

AuthConfig

Extension

mod_authn_dbm
<table>
<thead>
<tr>
<th>AuthDBMUserFile file-path</th>
<th>, .htaccess</th>
<th>AuthConfig</th>
<th>Extension</th>
<th>mod_authn_dbm</th>
</tr>
</thead>
</table>

| AuthDBMUserFile           | DBM         |

<table>
<thead>
<tr>
<th></th>
<th>apache module dbmopen NULL DBM</th>
<th>Netscape NULL</th>
</tr>
</thead>
</table>

|                      | Apache dbmmanage perl         |               |
Bugfix checklist

- httpd changelog
- Known issues
- Report a bug
- AuthBasicProvider
- AuthDigestProvider
- htpasswd
htdigest
AuthUserFile

ID

src/support

ht

ID username Filename :

htpasswd -c Filename username

Filename username2:

htpasswd Filename username2

():

HTTP htpasswd htdj

AuthUserFile
Apache Module mod_authn_socache

**Description:** Manages a cache of authentication credentials to relieve the load on backends

**Status:** Base

**Module Identifier:** authn_socache_module

**Source File:** mod_authn_socache.c

**Compatibility:** Version 2.3 and later

**Summary**

Maintains a cache of authentication credentials, so that a new backend lookup is not required for every authenticated request.
Some users of more heavyweight authentication such as SQL database lookups (mod_authn_dbd) have reported it putting an unacceptable load on their authentication provider. A typical case in point is where an HTML page contains hundreds of objects (images, scripts, stylesheets, media, etc), and a request to the page generates hundreds of effectively-immediate requests for authenticated additional contents.

mod_authn_socache provides a solution to this problem by maintaining a cache of authentication credentials.
The authentication cache should be used where authentication lookups impose a significant load on the server, or a backend or network. Authentication by file (mod_authn_file) or dbm (mod_authn_dbm) are unlikely to benefit, as these are fast and lightweight in their own right (though in some cases, such as a network-mounted file, caching may be worthwhile). Other providers such as SQL or LDAP based authentication are more likely to benefit, particularly where there is an observed performance issue. Amongst the standard modules, mod_authnz_ldap manages its own cache, so only mod_authn_dbd will usually benefit from this cache.

The basic rules to cache for a provider are:

1. Include the provider you're caching for in an AuthnCacheProvideFor directive.
2. List socache ahead of the provider you're caching for in your AuthBasicProvider or AuthDigestProvider directive.

A simple usage example to accelerate mod_authn_dbd using dbm as a cache engine:

```
#AuthnCacheSOCache is optional. If specified, AuthnCacheSOCache dbm
<Directory "/usr/www/myhost/private">
  AuthType Basic
  AuthName "Cached Authentication Example"
  AuthBasicProvider socache dbd
  AuthDBDUserPWQuery "SELECT password FROM authn WHERE user = %s"
  AuthnCacheProvideFor dbd
  Require valid-user
  #Optional
  AuthnCacheContext dbd-authn-example
</Directory>
```
Module developers should note that their modules must be enabled for cacheing with mod_authn_socache. A single optional API function `ap_authn_cache_store` is provided to cache credentials a provider has just looked up or generated. Usage examples are available in r957072, in which three authn providers are enabled for cacheing.
Description: Specify a context string for use in the cache key

Syntax: AuthnCacheContext
directory|server|custom-string

Default: directory

Context: directory

Status: Base

Module: mod_authn_socache

This directive specifies a string to be used along with the supplied username (and realm in the case of Digest Authentication) in constructing a cache key. This serves to disambiguate identical usernames serving different authentication areas on the server.

Two special values for this are directory, which uses the directory context of the request as a string, and server which uses the virtual host name.

The default is directory, which is also the most conservative setting. This is likely to be less than optimal, as it (for example) causes $app-base, $app-base/images, $app-base/scripts and $app-base/media each to have its own separate cache key. A better policy is to name the AuthnCacheContext for the password provider: for example a htpasswd file or database table.

Contexts can be shared across different areas of a server, where credentials are shared. However, this has potential to become a vector for cross-site or cross-application security breaches, so this directive is not permitted in .htaccess contexts.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Enable Authn caching configured anywhere</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>AuthnCacheEnable</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Override:</strong></td>
<td>None</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_authn_socache</td>
</tr>
</tbody>
</table>

This directive is not normally necessary: it is implied if authentication caching is enabled anywhere in `httpd.conf`. However, if it is not enabled anywhere in `httpd.conf` it will by default not be initialised, and is therefore not available in a `.htaccess` context. This directive ensures it is initialised so it can be used in `.htaccess`. 
**Description:** Specify which authn provider(s) to cache for

**Syntax:**

```
AuthnCacheProvideFor authn-provider [...]
```

**Default:** None

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_authn_socache

This directive specifies an authentication provider or providers to cache for. Credentials found by a provider not listed in an AuthnCacheProvideFor directive will not be cached.

For example, to cache credentials found by `mod_authn_dbd` or by a custom provider `myprovider`, but leave those looked up by lightweight providers like file or dbm lookup alone:

```
AuthnCacheProvideFor dbd myprovider
```
**Description:** Select socache backend provider to use

**Syntax:** AuthnCacheSOCache *provider-name*[:*provider-args*]

**Context:** server config

**Override:** None

**Status:** Base

**Module:** mod_authn_socache

**Compatibility:** Optional provider arguments are available in Apache HTTP Server 2.4.7 and later

This is a server-wide setting to select a provider for the shared object cache, followed by optional arguments for that provider. Some possible values for *provider-name* are "dbm", "dc", "memcache", or "shmcb", each subject to the appropriate module being loaded. If not set, your platform's default will be used.
**Description:** Set a timeout for cache entries

**Syntax:** AuthnCacheTimeout *timeout* (seconds)

**Default:** 300 (5 minutes)

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_authn_socache

Cacheing authentication data can be a security issue, though short-term cacheing is unlikely to be a problem. Typically a good solution is to cache credentials for as long as it takes to relieve the load on a backend, but no longer, though if changes to your users and passwords are infrequent then a longer timeout may suit you. The default 300 seconds (5 minutes) is both cautious and ample to keep the load on a backend such as dbd (SQL database queries) down.

This should not be confused with session timeout, which is an entirely separate issue. However, you may wish to check your session-management software for whether cached credentials can "accidentally" extend a session, and bear it in mind when setting your timeout.
Apache Module mod_authnz_fcgi

**Description:** Allows a FastCGI authorizer application to handle Apache httpd authentication and authorization

**Status:** Extension

**Module Identifier:** authnz_fcgi_module

**Source File:** mod_authnz_fcgi.c

**Compatibility:** Available in version 2.4.10 and later

**Summary**

This module allows FastCGI authorizer applications to authenticate users and authorize access to resources. It supports generic FastCGI authorizers which participate in a single phase for authentication and authorization as well as Apache httpd-specific authenticators and authorizors which participate in one or both phases.

FastCGI authorizers can authenticate using user id and password, such as for Basic authentication, or can authenticate using arbitrary mechanisms.

**Bugfix checklist**

- httpd changelog
- Known issues
- Report a bug

**See also**
Authentication, Authorization, and Access Control

mod_auth_basic
fcgistarter
mod_proxy_fcgi
The invocation modes for FastCGI authorizers supported by this module are distinguished by two characteristics, type and auth mechanism.

Type is simply authn for authentication, authz for authorization, or authnz for combined authentication and authorization.

Auth mechanism refers to the Apache httpd configuration mechanisms and processing phases, and can be AuthBasicProvider, Require, or check_user_id. The first two of these correspond to the directives used to enable participation in the appropriate processing phase.

Descriptions of each mode:

**Type authn, mechanism AuthBasicProvider**
In this mode, FCGI_ROLE is set to AUTHORIZER and FCGI_APACHE_ROLE is set to AUTHENTICATOR. The application must be defined as provider type authn using AuthnzFcgiDefineProvider and enabled with AuthBasicProvider. When invoked, the application is expected to authenticate the client using the provided user id and password. Example application:

```perl
#!/usr/bin/perl
use FCGI;
my $request = FCGI::Request();
while ($request->Accept() >= 0) {
    die if $ENV{'FCGI_APACHE_ROLE'} ne ""
    die if $ENV{'FCGI_ROLE'} ne ""
    die if !$ENV{'REMOTE_PASSWD'};
    die if !$ENV{'REMOTE_USER'};
```
print STDERR "This text is written to the web server error log.
"

if ( ($ENV{'REMOTE_USER'} eq "foo" 
       $ENV{'REMOTE_PASSWD'} eq "bar" ) 
     print "Status: 200\n"
     print "Variable-AUTHN_1: authn_01\n"
     print "Variable-AUTHN_2: authn_02\n"
     print "\n"
   } 
   else { 
     print "Status: 401\n\n"
   } 
"
Example configuration:

AuthnzFcgiDefineProvider authn FooAuthn
<Location "/protected/">
  AuthType Basic
  AuthName "Restricted"
  AuthBasicProvider FooAuthn
  Require ...
</Location>

**Type authz, mechanism Require**

In this mode, FCGI_ROLE is set to AUTHORIZER and FCGI_APACHE_ROLE is set to AUTHORIZER. The application must be defined as provider type authz using AuthnzFcgiDefineProvider. When invoked, the application is expected to authorize the client using the provided user id and other request data. Example application:
#!/usr/bin/perl
use FCGI;
my $request = FCGI::Request();
while ($request->Accept() >= 0) {
    die if $ENV{'FCGI_APACHE_ROLE'} ne "AUTHORIZER";
    die if $ENV{'FCGI_ROLE'} ne "AUTHORIZER";
    die if $ENV{'REMOTE_PASSWD'};

    print STDERR "This text is written to the web server error log."

    if ($ENV{'REMOTE_USER'} eq "foo1") {
        print "Status: 200
";
        print "Variable-AUTHZ_1: authz_01"
        print "Variable-AUTHZ_2: authz_02"
        print "\n"
    } else {
        print "Status: 403\n\n"
    }
}

Example configuration:

AuthnzFcgiDefineProvider authz FooAuthz
<Location "/protected"/>
    AuthType ...
    AuthName ...
    AuthBasicProvider ...
    Require FooAuthz
</Location>

*Type authnz, mechanism AuthBasicProvider + Require*
In this mode, which supports the web server-agnostic FastCGI AUTHORIZER protocol, FCGI_ROLE is set to AUTHORIZER and FCGI_APACHE_ROLE is not set. The application must be defined as provider type authnz using AuthnzFcgiDefineProvider. The application is expected to handle both authentication and authorization in the same invocation using the user id, password, and other request data. The invocation occurs during the Apache httpd API authentication phase. If the application returns 200 and the same provider is invoked during the authorization phase (via Require), mod_authnz_fcgi will return success for the authorization phase without invoking the application. Example application:

```perl
#!/usr/bin/perl
use FCGI;

my $request = FCGI::Request();
while ($request->Accept() >= 0) {
    die if $ENV{'FCGI_APACHE_ROLE'};
    die if $ENV{'FCGI_ROLE'} ne "AUTHORIZER";
    die if !$ENV{'REMOTE_PASSWD'};
    die if !$ENV{'REMOTE_USER'};

    print STDERR "This text is written to the web server error log."

    if ( ($ENV{'REMOTE_USER'} eq "foo" 
         || $ENV{'REMOTE_USER'} eq "foo1") 
         && $ENV{'REMOTE_PASSWD'} eq "bar" 
         && $ENV{'REQUEST_URI'} =~ m%/bar/.*%)
        print "Status: 200\n";
        print "Variable-AUTHNZ_1: authnz_01\n";
        print "Variable-AUTHNZ_2: authnz_02\n";
    }
}
```

Example configuration:

```xml
<AuthnzFcgiDefineProvider authnz FooAuthnz
<Location "/protected/"
  AuthType Basic
  AuthName "Restricted"
  AuthBasicProvider FooAuthnz
  Require FooAuthnz
</Location>
```

**Type authn, mechanism check_user_id**

In this mode, FCGI_ROLE is set to AUTHORIZER and FCGI_APACHE_ROLE is set to AUTHENTICATOR. The application must be defined as provider type `authn` using `AuthnzFcgiDefineProvider`. `AuthnzFcgiCheckAuthnProvider` specifies when it is called. Example application:

```perl
#!/usr/bin/perl
use FCGI;
my $request = FCGI::Request();
while ($request->Accept() >= 0) {
  die if $ENV{'FCGI_APACHE_ROLE'} ne "AUTHENTICATOR";
  die if $ENV{'FCGI_ROLE'} ne "AUTHORIZER";

  # This authorizer assumes that the RequireBasicAuth option of AuthnzFcgiCheckAuthnProvider is On:
  die if !$ENV{'REMOTE_PASSWD'};
```
die if !$ENV{'REMOTE_USER'};

print STDERR "This text is written to the web server error log."

if ( ($ENV{'REMOTE_USER'} eq "foo" && 
      $ENV{'REMOTE_PASSWD'} eq "bar")
   || ($ENV{'REMOTE_USER'} eq "foo1")
   && $ENV{'REMOTE_PASSWD'} eq "bar")
  print "Status: 200\n"
  print "Variable-AUTHNZ_1: authnz_01\n"
  print "Variable-AUTHNZ_2: authnz_02\n"
  print "\n"
else {
  print "Status: 401\n"
  # If a response body is written to the client.
}

Example configuration:

AuthnzFcgiDefineProvider authn FooAuthn
<Location "/protected/">
  AuthType ...
  AuthName ...
  AuthnzFcgiCheckAuthnProvider FooAuthn
  Authoritative On
  RequireBasicAuth Off
  UserExpr "%{reqenv:REMOTE_USER}"
  Require ...
</Location>
1. If your application supports the separate authentication and authorization roles (AUTHENTICATOR and AUTHORIZER), define separate providers as follows, even if they map to the same application:

```
AuthnzFcgiDefineProvider authn  FooAuthn
AuthnzFcgiDefineProvider authz  FooAuthz
```

Specify the authn provider on `AuthBasicProvider` and the authz provider on `Require`:

```
AuthType Basic
AuthName "Restricted"
AuthBasicProvider FooAuthn
Require FooAuthz
```

2. If your application supports the generic AUTHORIZER role (authentication and authorizer in one invocation), define a single provider as follows:

```
AuthnzFcgiDefineProvider authnz FooAuthnz
```

Specify the authnz provider on both `AuthBasicProvider` and `Require`:

```
AuthType Basic
AuthName "Restricted"
AuthBasicProvider FooAuthnz
Require FooAuthnz
```
The following are potential features which are not currently implemented:

**Apache httpd access checker**

The Apache httpd API *access check* phase is a separate phase from authentication and authorization. Some other FastCGI implementations implement this phase, which is denoted by the setting of FCGI_APACHE_ROLE to ACCESS_CHECKER.

**Local (Unix) sockets or pipes**

Only TCP sockets are currently supported.

**Support for mod_authn_socache**

mod_authn_socache interaction should be implemented for applications which participate in Apache httpd-style authentication.

**Support for digest authentication using AuthDigestProvider**

This is expected to be a permanent limitation as there is no authorizer flow for retrieving a hash.

**Application process management**

This is expected to be permanently out of scope for this module. Application processes must be controlled by other means. For example, `fcgistarter` can be used to start them.

**AP_AUTH_INTERNAL_PER_URI**

All providers are currently registered as AP_AUTH_INTERNAL_PER_CONF, which means that checks are not performed again for internal subrequests with the same access control configuration as the initial request.

**Protocol data charset conversion**

If mod_authnz_fcgii runs in an EBCDIC compilation environment, all FastCGI protocol data is written in EBCDIC
and expected to be received in EBCDIC.

**Multiple requests per connection**
Currently the connection to the FastCGI authorizer is closed after every phase of processing. For example, if the authorizer handles separate *authn* and *authz* phases then two connections will be used.

**URI Mapping**
URIs from clients can't be mapped, such as with the *ProxyPass* used with FastCGI responders.
1. Processing errors are logged at log level `error` and higher.

2. Messages written by the application are logged at log level `warn`.

3. General messages for debugging are logged at log level `debug`.

4. Environment variables passed to the application are logged at log level `trace2`. The value of the `REMOTE_PASSWD` variable will be obscured, but **any other sensitive data will be visible in the log**.

5. All I/O between the module and the FastCGI application, including all environment variables, will be logged in printable and hex format at log level `trace5`. **All sensitive data will be visible in the log.**

`LogLevel` can be used to configure a log level specific to `mod_authnz_fcgii`. For example:

```
LogLevel info authnz_fcgii:trace8
```
**Description:** Enables a FastCGI application to handle the check_authn authentication hook.

**Syntax:**

```
AuthnzFcgiCheckAuthnProvider
provider-name|None option ...
```

**Default:** none

**Context:** directory

**Status:** Extension

**Module:** mod_authnz_fcgi

This directive is used to enable a FastCGI authorizer to handle a specific processing phase of authentication or authorization.

Some capabilities of FastCGI authorizers require enablement using this directive instead of **AuthBasicProvider**:

- Non-Basic authentication; generally, determining the user id of the client and returning it from the authorizer; see the UserExpr option below
- Selecting a custom response code; for a non-200 response from the authorizer, the code from the authorizer will be the status of the response
- Setting the body of a non-200 response; if the authorizer provides a response body with a non-200 response, that body will be returned to the client; up to 8192 bytes of text are supported

**provider-name**

This is the name of a provider defined with **AuthnzFcgiDefineProvider**.

**None**

Specify None to disable a provider enabled with this directive in an outer scope, such as in a parent directory.
The following options are supported:

**Authoritative On|Off (default On)**
This controls whether or not other modules are allowed to run when this module has a FastCGI authorizer configured and it fails the request.

**DefaultUser userid**
When the authorizer returns success and UserExpr is configured and evaluates to an empty string (e.g., authorizer didn't return a variable), this value will be used as the user id. This is typically used when the authorizer has a concept of guest, or unauthenticated, users and guest users are mapped to some specific user id for logging and other purposes.

**RequireBasicAuth On|Off (default Off)**
This controls whether or not Basic auth is required before passing the request to the authorizer. If required, the authorizer won't be invoked without a user id and password; 401 will be returned for a request without that.

**UserExpr expr (no default)**
When Basic authentication isn't provided by the client and the authorizer determines the user, this expression, evaluated after calling the authorizer, determines the user. The expression follows [ap_expr syntax](https://httpd.apache.org/docs/2.4/rewrite/rewrite.html#ap_expr) and must resolve to a string. A typical use is to reference a Variable-XXX setting returned by the authorizer using an option like `UserExpr "%{reqenv:XXX}"`. If this option is specified and the user id can't be retrieved using the expression after a successful authentication, the request will be rejected with a 500 error.
**Description:** Defines a FastCGI application as a provider for authentication and/or authorization

**Syntax:**
```
AuthnzFcgiDefineProvider type provider-name backend-address
```

**Default:** none

**Context:** server config

**Status:** Extension

**Module:** mod_authnz_fcgi

This directive is used to define a FastCGI application as a provider for a particular phase of authentication or authorization.

**type**
This must be set to `authn` for authentication, `authz` for authorization, or `authnz` for a generic FastCGI authorizer which performs both checks.

**provider-name**
This is used to assign a name to the provider which is used in other directives such as `AuthBasicProvider` and `Require`.

**backend-address**
This specifies the address of the application, in the form `fcgi://hostname:port/`. The application process(es) must be managed independently, such as with `fcgistarter`.
Apache Module mod_authnz_ldap

**Description:** Allows an LDAP directory to be used to store the database for HTTP Basic authentication.

**Status:** Extension

**Module Identifier:** authnz_ldap_module

**Source File:** mod_authnz_ldap.c

**Compatibility:** Available in version 2.1 and later

---

**Summary**

This module allows authentication front-ends such as mod_auth_basic to authenticate users through an ldap directory.

mod_authnz_ldap supports the following features:

- Known to support the OpenLDAP SDK (both 1.x and 2.x), Novell LDAP SDK and the iPlanet (Netscape) SDK.
- Complex authorization policies can be implemented by representing the policy with LDAP filters.
- Uses extensive caching of LDAP operations via mod_ldap.
- Support for LDAP over SSL (requires the Netscape SDK) or TLS (requires the OpenLDAP 2.x SDK or Novell LDAP SDK).

When using mod_auth_basic, this module is invoked via the AuthBasicProvider directive with the ldap value.

---

**Bugfix checklist**
See also

- mod_ldap
- mod_auth_basic
- mod_authz_user
- mod_authz_groupfile
• General caveats
• Operation
  ■ The Authentication Phase
  ■ The Authorization Phase
• The Require Directives
  ■ Require ldap-user
  ■ Require ldap-group
  ■ Require ldap-dn
  ■ Require ldap-attribute
  ■ Require ldap-filter
• Examples
• Using TLS
• Using SSL
• Exposing Login Information
• Using Active Directory
• Using Microsoft FrontPage with mod_authnz_ldap
  ■ How It Works
  ■ Caveats
This module caches authentication and authorization results based on the configuration of `mod_ldap`. Changes made to the backing LDAP server will not be immediately reflected on the HTTP Server, including but not limited to user lockouts/revocations, password changes, or changes to group memberships. Consult the directives in `mod_ldap` for details of the cache tunables.
There are two phases in granting access to a user. The first phase is authentication, in which the `mod_authnz_ldap` authentication provider verifies that the user's credentials are valid. This is also called the *search/bind* phase. The second phase is authorization, in which `mod_authnz_ldap` determines if the authenticated user is allowed access to the resource in question. This is also known as the *compare* phase.

`mod_authnz_ldap` registers both an authn_ldap authentication provider and an authz_ldap authorization handler. The authn_ldap authentication provider can be enabled through the `AuthBasicProvider` directive using the `ldap` value. The authz_ldap handler extends the `Require` directive's authorization types by adding `ldap-user`, `ldap-dn` and `ldap-group` values.

### The Authentication Phase

During the authentication phase, `mod_authnz_ldap` searches for an entry in the directory that matches the username that the HTTP client passes. If a single unique match is found, then `mod_authnz_ldap` attempts to bind to the directory server using the DN of the entry plus the password provided by the HTTP client. Because it does a search, then a bind, it is often referred to as the search/bind phase. Here are the steps taken during the search/bind phase.

1. Generate a search filter by combining the attribute and filter provided in the `AuthLDAPURL` directive with the username passed by the HTTP client.
2. Search the directory using the generated filter. If the search does not return exactly one entry, deny or decline access.
3. Fetch the distinguished name of the entry retrieved from the search and attempt to bind to the LDAP server using that DN
and the password passed by the HTTP client. If the bind is unsuccessful, deny or decline access.

The following directives are used during the search/bind phase

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthLDAPURL</td>
<td>Specifies the LDAP server, the base DN, the attribute to use in the search, as well as the extra search filter to use.</td>
</tr>
<tr>
<td>AuthLDAPBindDN</td>
<td>An optional DN to bind with during the search phase.</td>
</tr>
<tr>
<td>AuthLDAPBindPassword</td>
<td>An optional password to bind with during the search phase.</td>
</tr>
</tbody>
</table>

**The Authorization Phase**

During the authorization phase, mod_authnz_ldap attempts to determine if the user is authorized to access the resource. Many of these checks require mod_authnz_ldap to do a compare operation on the LDAP server. This is why this phase is often referred to as the compare phase. mod_authnz_ldap accepts the following Require directives to determine if the credentials are acceptable:

- Grant access if there is a Require ldap-user directive, and the username in the directive matches the username passed by the client.
- Grant access if there is a Require ldap-dn directive, and the DN in the directive matches the DN fetched from the LDAP directory.
- Grant access if there is a Require ldap-group directive, and the DN fetched from the LDAP directory (or the username passed by the client) occurs in the LDAP group or, potentially, in one of its sub-groups.
Grant access if there is a `Require ldap-rewrite` directive, and the attribute fetched from the LDAP directory matches the given value.

Grant access if there is a `Require ldap-filter` directive, and the search filter successfully finds a single user object that matches the DN of the authenticated user.

Otherwise, deny or decline access.

Other `Require` values may also be used which may require loading additional authorization modules.

- Grant access to all successfully authenticated users if there is a `Require valid-user` directive. (requires `mod_authz_user`)

- Grant access if there is a `Require group` directive, and `mod_authz_groupfile` has been loaded with the `AuthGroupFile` directive set.

- Others...

`mod_authnz_ldap` uses the following directives during the compare phase:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthLDAPURL</td>
<td>The attribute specified in the URL is used in compare operations for the Require ldap-user operation.</td>
</tr>
<tr>
<td>AuthLDAPCompareDNOnServer</td>
<td>Determines the behavior of the Require ldap-dn directive.</td>
</tr>
<tr>
<td>AuthLDAPGroupAttribute</td>
<td>Determines the attribute to use for comparisons in the Require ldap-group directive.</td>
</tr>
<tr>
<td>AuthLDAPGroupAttributeIsDN</td>
<td>Specifies whether to use the</td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AuthLDAPMaxSubGroupDepth</td>
<td>Determines the maximum depth of sub-groups that will be evaluated during comparisons in the <code>Require ldap-group</code> directive.</td>
</tr>
<tr>
<td>AuthLDAPSubGroupAttribute</td>
<td>Determines the attribute to use when obtaining sub-group members of the current group during comparisons in the <code>Require ldap-group</code> directive.</td>
</tr>
<tr>
<td>AuthLDAPSubGroupClass</td>
<td>Specifies the LDAP objectClass values used to identify if queried directory objects really are group objects (as opposed to user objects) during the <code>Require ldap-group</code> directive's sub-group processing.</td>
</tr>
</tbody>
</table>
Apache's `Require` directives are used during the authorization phase to ensure that a user is allowed to access a resource. `mod_authnz_ldap` extends the authorization types with `ldap-user`, `ldap-dn`, `ldap-group`, `ldap-attribute` and `ldap-filter`. Other authorization types may also be used but may require that additional authorization modules be loaded.

Since v2.4.8, `expressions` are supported within the LDAP require directives.

**Require ldap-user**

The `Require ldap-user` directive specifies what usernames can access the resource. Once `mod_authnz_ldap` has retrieved a unique DN from the directory, it does an LDAP compare operation using the username specified in the `Require ldap-user` to see if that username is part of the just-fetched LDAP entry. Multiple users can be granted access by putting multiple usernames on the line, separated with spaces. If a username has a space in it, then it must be surrounded with double quotes. Multiple users can also be granted access by using multiple `Require ldap-user` directives, with one user per line. For example, with a `AuthLDAPURL` of `ldap://ldap/o=Example?cn` (i.e., `cn` is used for searches), the following `Require` directives could be used to restrict access:

```
Require ldap-user "Barbara Jenson"
Require ldap-user "Fred User"
Require ldap-user "Joe Manager"
```

Because of the way that `mod_authnz_ldap` handles this directive, Barbara Jenson could sign on as `Barbara Jenson`, `Babs Jenson` or any other `cn` that she has in her LDAP entry. Only the
single Require ldap-user line is needed to support all values of the attribute in the user's entry.

If the uid attribute was used instead of the cn attribute in the URL above, the above three lines could be condensed to

```
Require ldap-user bjenson fuser jmanager
```

**Require ldap-group**

This directive specifies an LDAP group whose members are allowed access. It takes the distinguished name of the LDAP group. Note: Do not surround the group name with quotes. For example, assume that the following entry existed in the LDAP directory:

```
dn: cn=Administrators, o=Example
objectClass: groupOfUniqueNames
uniqueMember: cn=Barbara Jenson, o=Example
uniqueMember: cn=Fred User, o=Example
```

The following directive would grant access to both Fred and Barbara:

```
Require ldap-group cn=Administrators, o=Example
```

Members can also be found within sub-groups of a specified LDAP group if **AuthLDAPMaxSubGroupDepth** is set to a value greater than 0. For example, assume the following entries exist in the LDAP directory:

```
dn: cn=Employees, o=Example
objectClass: groupOfUniqueNames
uniqueMember: cn=Managers, o=Example
uniqueMember: cn=Administrators, o=Example
uniqueMember: cn=Users, o=Example
```
The following directives would allow access for Bob Ellis, Tom Jackson, Barbara Jenson, Fred User, Allan Jefferson, and Paul Tilley but would not allow access for Jim Swenson, or Elliot Rhodes (since they are at a sub-group depth of 2):

```
Require ldap-group cn=Employees, o=Example
AuthLDAPMaxSubGroupDepth 1
```

Behavior of this directive is modified by the 
- AuthLDAPGroupAttribute,
- AuthLDAPGroupAttributeIsDN,
- AuthLDAPMaxSubGroupDepth,
- AuthLDAPSubGroupAttribute, and
- AuthLDAPSubGroupClass directives.

**Require ldap-dn**

The `Require ldap-dn` directive allows the administrator to grant
access based on distinguished names. It specifies a DN that must match for access to be granted. If the distinguished name that was retrieved from the directory server matches the distinguished name in the Require ldap-dn, then authorization is granted. Note: do not surround the distinguished name with quotes.

The following directive would grant access to a specific DN:

```
Require ldap-dn cn=Barbara Jenson, o=Example
```

Behavior of this directive is modified by the `AuthLDAPCompareDNOnServer` directive.

**Require ldap-attribute**

The Require ldap-attribute directive allows the administrator to grant access based on attributes of the authenticated user in the LDAP directory. If the attribute in the directory matches the value given in the configuration, access is granted.

The following directive would grant access to anyone with the attribute `employeeType = active`

```
Require ldap-attribute "employeeType=active"
```

Multiple attribute/value pairs can be specified on the same line separated by spaces or they can be specified in multiple Require ldap-attribute directives. The effect of listing multiple attribute/values pairs is an OR operation. Access will be granted if any of the listed attribute values match the value of the corresponding attribute in the user object. If the value of the attribute contains a space, only the value must be within double
quotes.

The following directive would grant access to anyone with the city attribute equal to "San Jose" or status equal to "Active"

```
Require ldap-attribute city="San Jose" status=active
```

### Require ldap-filter

The Require ldap-filter directive allows the administrator to grant access based on a complex LDAP search filter. If the dn returned by the filter search matches the authenticated user dn, access is granted.

The following directive would grant access to anyone having a cell phone and is in the marketing department

```
Require ldap-filter 
&(cell=*)(department=marketing)
```

The difference between the Require ldap-filter directive and the Require ldap-attribute directive is that ldap-filter performs a search operation on the LDAP directory using the specified search filter rather than a simple attribute comparison. If a simple attribute comparison is all that is required, the comparison operation performed by ldap-attribute will be faster than the search operation used by ldap-filter especially within a large directory.
- Grant access to anyone who exists in the LDAP directory, using their UID for searches.

```
AuthLDAPURL "ldap://ldap.example.com:389/ou=People,o=Example?uid?sub?(objectClass=*)
Require valid-user
```

- The next example is the same as above; but with the fields that have useful defaults omitted. Also, note the use of a redundant LDAP server.

```
AuthLDAPURL "ldap://ldap1.example.com ldap2.example.com/ou=People,o=Example"
Require valid-user
```

- The next example is similar to the previous one, but it uses the common name instead of the UID. Note that this could be problematical if multiple people in the directory share the same cn, because a search on cn must return exactly one entry. That's why this approach is not recommended: it's a better idea to choose an attribute that is guaranteed unique in your directory, such as uid.

```
AuthLDAPURL "ldap://ldap.example.com/ou=People,o=Example?cn"
Require valid-user
```

- Grant access to anybody in the Administrators group. The users must authenticate using their UID.

```
AuthLDAPURL ldap://ldap.example.com/o=Example?uid
Require ldap-group cn=Administrators, o=
```
Grant access to anybody in the group whose name matches the hostname of the virtual host. In this example an expression is used to build the filter.

```
AuthLDAPURL ldap://ldap.example.com/o=Example?uid
Require ldap-group cn=%{SERVER_NAME}, o=Example
```

The next example assumes that everyone at Example who carries an alphanumeric pager will have an LDAP attribute of qpagePagerID. The example will grant access only to people (authenticated via their UID) who have alphanumeric pagers:

```
AuthLDAPURL ldap://ldap.example.com/o=Example?uid??(qpagePagerID=*)
Require valid-user
```

The next example demonstrates the power of using filters to accomplish complicated administrative requirements. Without filters, it would have been necessary to create a new LDAP group and ensure that the group's members remain synchronized with the pager users. This becomes trivial with filters. The goal is to grant access to anyone who has a pager, plus grant access to Joe Manager, who doesn't have a pager, but does need to access the same resource:

```
AuthLDAPURL ldap://ldap.example.com/o=Example?uid??(|(qpagePagerID=*)(uid=jmanager))
Require valid-user
```

This last may look confusing at first, so it helps to evaluate what the search filter will look like based on who connects, as shown below. If Fred User connects as fuser, the filter would
look like

\[(\text{qpagePagerID} \neq *) \land (\text{uid} = \text{jmanager}) \land (\text{uid} = \text{fuser})\]

The above search will only succeed if \textit{fuser} has a pager. When Joe Manager connects as \textit{jmanager}, the filter looks like

\[(\text{qpagePagerID} \neq *) \land (\text{uid} = \text{jmanager}) \land (\text{uid} = \text{jmanager})\]

The above search will succeed whether \textit{jmanager} has a pager or not.
To use TLS, see the `mod_ldap` directives `LDAPTrustedClientCert`, `LDAPTrustedGlobalCert` and `LDAPTrustedMode`.

An optional second parameter can be added to the `AuthLDAPURL` to override the default connection type set by `LDAPTrustedMode`. This will allow the connection established by an `ldap://` URL to be upgraded to a secure connection on the same port.
To use SSL, see the `mod_ldap` directives `LDAPTrustedClientCert`, `LDAPTrustedGlobalCert` and `LDAPTrustedMode`.

To specify a secure LDAP server, use `ldaps://` in the `AuthLDAPURL` directive, instead of `ldap://`. 
when this module performs authentication, ldap attributes specified in the authldapurl directive are placed in environment variables with the prefix "AUTHENTICATE_".

when this module performs authorization, ldap attributes specified in the authldapurl directive are placed in environment variables with the prefix "AUTHORIZE_".

If the attribute field contains the username, common name and telephone number of a user, a CGI program will have access to this information without the need to make a second independent LDAP query to gather this additional information.

This has the potential to dramatically simplify the coding and configuration required in some web applications.
An Active Directory installation may support multiple domains at the same time. To distinguish users between domains, an identifier called a User Principle Name (UPN) can be added to a user's entry in the directory. This UPN usually takes the form of the user's account name, followed by the domain components of the particular domain, for example somebody@nz.example.com.

You may wish to configure the `mod_authnz_ldap` module to authenticate users present in any of the domains making up the Active Directory forest. In this way both somebody@nz.example.com and someone@au.example.com can be authenticated using the same query at the same time.

To make this practical, Active Directory supports the concept of a Global Catalog. This Global Catalog is a read only copy of selected attributes of all the Active Directory servers within the Active Directory forest. Querying the Global Catalog allows all the domains to be queried in a single query, without the query spanning servers over potentially slow links.

If enabled, the Global Catalog is an independent directory server that runs on port 3268 (3269 for SSL). To search for a user, do a subtree search for the attribute `userPrincipalName`, with an empty search root, like so:

```bash
AuthLDAPBindDN apache@example.com
AuthLDAPBindPassword password
AuthLDAPURL ldap://10.0.0.1:3268/?userPrincipalName
```

Users will need to enter their User Principal Name as a login, in the form somebody@nz.example.com.
Normally, FrontPage uses FrontPage-web-specific user/group files (i.e., the `mod_authn_file` and `mod_authz_groupfile` modules) to handle all authentication. Unfortunately, it is not possible to just change to LDAP authentication by adding the proper directives, because it will break the `Permissions` forms in the FrontPage client, which attempt to modify the standard text-based authorization files.

Once a FrontPage web has been created, adding LDAP authentication to it is a matter of adding the following directives to every `.htaccess` file that gets created in the web:

<table>
<thead>
<tr>
<th>Directive</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuthLDAPURL</td>
<td>&quot;the url&quot;</td>
</tr>
<tr>
<td>AuthGroupFile</td>
<td>&quot;mygroupfile&quot;</td>
</tr>
<tr>
<td>Require group</td>
<td>&quot;mygroupfile&quot;</td>
</tr>
</tbody>
</table>

**How It Works**

FrontPage restricts access to a web by adding the `Require valid-user` directive to the `.htaccess` files. The `Require valid-user` directive will succeed for any user who is valid as far as LDAP is concerned. This means that anybody who has an entry in the LDAP directory is considered a valid user, whereas FrontPage considers only those people in the local user file to be valid. By substituting the ldap-group with group file authorization, Apache is allowed to consult the local user file (which is managed by FrontPage) - instead of LDAP - when handling authorizing the user.

Once directives have been added as specified above, FrontPage users will be able to perform all management operations from the FrontPage client.
Caveats

- When choosing the LDAP URL, the attribute to use for authentication should be something that will also be valid for putting into a `mod_authn_file` user file. The user ID is ideal for this.
- When adding users via FrontPage, FrontPage administrators should choose usernames that already exist in the LDAP directory (for obvious reasons). Also, the password that the administrator enters into the form is ignored, since Apache will actually be authenticating against the password in the LDAP database, and not against the password in the local user file. This could cause confusion for web administrators.
- Apache must be compiled with `mod_auth_basic`, `mod_authn_file` and `mod_authz_groupfile` in order to use FrontPage support. This is because Apache will still use the `mod_authz_groupfile` group file for determine the extent of a user's access to the FrontPage web.
- The directives must be put in the `.htaccess` files. Attempting to put them inside `<Location>` or `<Directory>` directives won't work. This is because `mod_authnz_ldap` has to be able to grab the `AuthGroupFile` directive that is found in FrontPage `.htaccess` files so that it knows where to look for the valid user list. If the `mod_authnz_ldap` directives aren't in the same `.htaccess` file as the FrontPage directives, then the hack won't work, because `mod_authnz_ldap` will never get a chance to process the `.htaccess` file, and won't be able to find the FrontPage-managed user file.
**Description:** Specifies the prefix for environment variables set during authorization

**Syntax:**

```
AuthLDAPAuthorizePrefix prefix
```

**Default:**

```
AuthLDAPAuthorizePrefix AUTHORIZE_
```

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

**Compatibility:** Available in version 2.3.6 and later

This directive allows you to override the prefix used for environment variables set during LDAP authorization. If AUTHENTICATE_ is specified, consumers of these environment variables see the same information whether LDAP has performed authentication, authorization, or both.

**Note**

No authorization variables are set when a user is authorized on the basis of Require valid-user.
**Description:** Determines if other authentication providers are used when a user can be mapped to a DN but the server cannot successfully bind with the user's credentials.

**Syntax:** AuthLDAPBindAuthoritative off|on

**Default:** AuthLDAPBindAuthoritative on

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

By default, subsequent authentication providers are only queried if a user cannot be mapped to a DN, but not if the user can be mapped to a DN and their password cannot be verified with an LDAP bind. If AuthLDAPBindAuthoritative is set to off, other configured authentication modules will have a chance to validate the user if the LDAP bind (with the current user's credentials) fails for any reason.

This allows users present in both LDAP and AuthUserFile to authenticate when the LDAP server is available but the user's account is locked or password is otherwise unusable.

**See also**

- [AuthUserFile](#)
- [AuthBasicProvider](#)
An optional DN used to bind to the server when searching for entries. If not provided, mod_authnz_ldap will use an anonymous bind.
**Description:** Password used in conjunction with the bind DN

**Syntax:** `AuthLDAPBindPassword` `password`

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

**Compatibility:** exec: was added in 2.4.5.

A bind password to use in conjunction with the bind DN. Note that the bind password is probably sensitive data, and should be properly protected. You should only use the `AuthLDAPBindDN` and `AuthLDAPBindPassword` if you absolutely need them to search the directory.

If the value begins with exec: the resulting command will be executed and the first line returned to standard output by the program will be used as the password.

```plaintext
#Password used as-is
AuthLDAPBindPassword secret

#Run /path/to/program to get my password
AuthLDAPBindPassword exec:/path/to/program

#Run /path/to/otherProgram and provide arguments
AuthLDAPBindPassword "exec:/path/to/otherProgram argument1"
```
**Description:** Language to charset conversion configuration file

**Syntax:** `AuthLDAPCharsetConfig` *file-path*

**Context:** server config

**Status:** Extension

**Module:** `mod_authnz_ldap`

The `AuthLDAPCharsetConfig` directive sets the location of the language to charset conversion configuration file. *File-path* is relative to the `ServerRoot`. This file specifies the list of language extensions to character sets. Most administrators use the provided `charset.conv` file, which associates common language extensions to character sets.

The file contains lines in the following format:

```
Language-Extension charset [Language-String] ...
```

The case of the extension does not matter. Blank lines, and lines beginning with a hash character (#) are ignored.
**Description:** Use the authenticated user's credentials to perform authorization comparisons

**Syntax:** AuthLDAPCompareAsUser on|off

**Default:** AuthLDAPCompareAsUser off

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

**Compatibility:** Available in version 2.3.6 and later

When set, and `mod_authnz_ldap` has authenticated the user, LDAP comparisons for authorization use the queried distinguished name (DN) and HTTP basic authentication password of the authenticated user instead of the servers configured credentials.

The `ldap-attribute`, `ldap-user`, and `ldap-group` (single-level only) authorization checks use comparisons.

This directive only has effect on the comparisons performed during nested group processing when [AuthLDAPSearchAsUser](#) is also enabled.

This directive should only be used when your LDAP server doesn't accept anonymous comparisons and you cannot use a dedicated [AuthLDAPBindDN](#).

**See also**

- [AuthLDAPInitialBindAsUser](#)
- [AuthLDAPSearchAsUser](#)
**Description:** Use the LDAP server to compare the DNs

**Syntax:** AuthLDAPCompareDNOnServer on|off

**Default:** AuthLDAPCompareDNOnServer on

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

When set, mod_authnz_ldap will use the LDAP server to compare the DNs. This is the only foolproof way to compare DNs. mod_authnz_ldap will search the directory for the DN specified with the Require dn directive, then, retrieve the DN and compare it with the DN retrieved from the user entry. If this directive is not set, mod_authnz_ldap simply does a string comparison. It is possible to get false negatives with this approach, but it is much faster. Note the mod_ldap cache can speed up DN comparison in most situations.
**Description**: When will the module de-reference aliases

**Syntax**: AuthLDAPDereferenceAliases
never|searching|finding|always

**Default**: AuthLDAPDereferenceAliases always

**Context**: directory, .htaccess

**Override**: AuthConfig

**Status**: Extension

**Module**: mod_authnz_ldap

This directive specifies when mod_authnz_ldap will de-reference aliases during LDAP operations. The default is always.
**Description:** LDAP attributes used to identify the user members of groups.

**Syntax:** AuthLDAPGroupAttribute *attribute*

**Default:** AuthLDAPGroupAttribute member uniquemember

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

This directive specifies which LDAP attributes are used to check for user members within groups. Multiple attributes can be used by specifying this directive multiple times. If not specified, then `mod_authnz_ldap` uses the member and uniquemember attributes.
When set on, this directive says to use the distinguished name of the client username when checking for group membership. Otherwise, the username will be used. For example, assume that the client sent the username bjenson, which corresponds to the LDAP DN cn=Babs Jenson, o=Example. If this directive is set, `mod_authnz_ldap` will check if the group has cn=Babs Jenson, o=Example as a member. If this directive is not set, then `mod_authnz_ldap` will check if the group has bjenson as a member.
**Description:** Determines if the server does the initial DN lookup using the basic authentication users' own username, instead of anonymously or with hard-coded credentials for the server.

**Syntax:**
```
AuthLDAPInitialBindAsUser off|on
```

**Default:**
```
AuthLDAPInitialBindAsUser off
```

**Context:**
directory, .htaccess

**Override:**
AuthConfig

**Status:**
Extension

**Module:**
mod_authnz_ldap

**Compatibility:** Available in version 2.3.6 and later

By default, the server either anonymously, or with a dedicated user and password, converts the basic authentication username into an LDAP distinguished name (DN). This directive forces the server to use the verbatim username and password provided by the incoming user to perform the initial DN search.

If the verbatim username can't directly bind, but needs some cosmetic transformation, see `AuthLDAPInitialBindPattern`.

This directive should only be used when your LDAP server doesn't accept anonymous searches and you cannot use a dedicated `AuthLDAPBindDN`.

**Not available with authorization-only**

This directive can only be used if this module authenticates the user, and has no effect when this module is used exclusively for authorization.

**See also**
- AuthLDAPInitialBindPattern
- AuthLDAPBindDN
- AuthLDAPCompareAsUser
- AuthLDAPSearchAsUser
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Specifies the transformation of the basic authentication username to be used when binding to the LDAP server to perform a DN lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>AuthLDAPInitialBindPattern <em>regex substitution</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>AuthLDAPInitialBindPattern (.*) $1 (remote username used verbatim)</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>directory, .htaccess</td>
</tr>
<tr>
<td><strong>Override:</strong></td>
<td>AuthConfig</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_authnz_ldap</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in version 2.3.6 and later</td>
</tr>
</tbody>
</table>

If `AuthLDAPInitialBindAsUser` is set to `ON`, the basic authentication username will be transformed according to the regular expression and substitution arguments.

The regular expression argument is compared against the current basic authentication username. The substitution argument may contain backreferences, but has no other variable interpolation.

This directive should only be used when your LDAP server doesn't accept anonymous searches and you cannot use a dedicated `AuthLDAPBindDN`.

```
AuthLDAPInitialBindPattern (.+) $1@example.com
```

```
AuthLDAPInitialBindPattern (.+) cn=$1,dc=example,dc=com
```
**Not available with authorization-only**

This directive can only be used if this module authenticates the user, and has no effect when this module is used exclusively for authorization.

**debugging**

The substituted DN is recorded in the environment variable `LDAP_BINDASUSER`. If the regular expression does not match the input, the verbatim username is used.

**See also**

- [AuthLDAPInitialBindAsUser](#)
- [AuthLDAPBindDN](#)
### Description:
Specifies the maximum sub-group nesting depth that will be evaluated before the user search is discontinued.

### Syntax:
```
AuthLDAPMaxSubGroupDepth Number
```

### Default:
```
AuthLDAPMaxSubGroupDepth 10
```

### Context:
directory, .htaccess

### Override:
AuthConfig

### Status:
Extension

### Module:
mod_authnz_ldap

### Compatibility:
Available in version 2.3.0 and later

When this directive is set to a non-zero value $X$ combined with use of the `Require ldap-group someGroupDN` directive, the provided user credentials will be searched for as a member of the `someGroupDN` directory object or of any group member of the current group up to the maximum nesting level $X$ specified by this directive.

See the [Require ldap-group](#) section for a more detailed example.

### Nested groups performance

When `AuthLDAPSubGroupAttribute` overlaps with `AuthLDAPGroupAttribute` (as it does by default and as required by common LDAP schemas), uncached searching for subgroups in large groups can be very slow. If you use large, non-nested groups, set `AuthLDAPMaxSubGroupDepth` to zero.
**Description:** Use the value of the attribute returned during the user query to set the REMOTE_USER environment variable

**Syntax:** AuthLDAPRemoteUserAttribute uid

**Default:** none

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

If this directive is set, the value of the REMOTE_USER environment variable will be set to the value of the attribute specified. Make sure that this attribute is included in the list of attributes in the AuthLDAPUrl definition, otherwise this directive will have no effect. This directive, if present, takes precedence over AuthLDAPRemoteUserIsDN. This directive is useful should you want people to log into a website using an email address, but a backend application expects the username as a userid.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Use the DN of the client username to set the REMOTE_USER environment variable</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>AuthLDAPRemoteUserIsDN on</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>AuthLDAPRemoteUserIsDN off</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>directory, .htaccess</td>
</tr>
<tr>
<td><strong>Override:</strong></td>
<td>AuthConfig</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_authnz_ldap</td>
</tr>
</tbody>
</table>

If this directive is set to on, the value of the REMOTE_USER environment variable will be set to the full distinguished name of the authenticated user, rather than just the username that was passed by the client. It is turned off by default.
<table>
<thead>
<tr>
<th>Description:</th>
<th>Use the authenticated user's credentials to perform authorization searches</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>AuthLDAPSearchAsUser on</td>
</tr>
<tr>
<td>Default:</td>
<td>AuthLDAPSearchAsUser off</td>
</tr>
<tr>
<td>Context:</td>
<td>directory, .htaccess</td>
</tr>
<tr>
<td>Override:</td>
<td>AuthConfig</td>
</tr>
<tr>
<td>Status:</td>
<td>Extension</td>
</tr>
<tr>
<td>Module:</td>
<td>mod_authnz_ldap</td>
</tr>
<tr>
<td>Compatibility:</td>
<td>Available in version 2.3.6 and later</td>
</tr>
</tbody>
</table>

When set, and mod_authnz_ldap has authenticated the user, LDAP searches for authorization use the queried distinguished name (DN) and HTTP basic authentication password of the authenticated user instead of the servers configured credentials.

The ldap-filter and ldap-dn authorization checks use searches.

This directive only has effect on the comparisons performed during nested group processing when AuthLDAPCompareAsUser is also enabled.

This directive should only be used when your LDAP server doesn't accept anonymous searches and you cannot use a dedicated AuthLDAPBindDN.

See also

- AuthLDAPInitialBindAsUser
- AuthLDAPCompareAsUser
**Description:** Specifies the attribute labels, one value per directive line, used to distinguish the members of the current group that are groups.

**Syntax:** AuthLDAPSubGroupAttribute *attribute*

**Default:** AuthLDAPSubgroupAttribute member uniquemember

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

**Compatibility:** Available in version 2.3.0 and later

An LDAP group object may contain members that are users and members that are groups (called nested or sub groups). The AuthLDAPSubGroupAttribute directive identifies the labels of group members and the AuthLDAPGroupAttribute directive identifies the labels of the user members. Multiple attributes can be used by specifying this directive multiple times. If not specified, then mod_authnz_ldap uses the member and uniqueMember attributes.
### Description:
Specifies which LDAP objectClass values identify directory objects that are groups during subgroup processing.

### Syntax:
```
AuthLDAPSubGroupClass
LdapObjectClass
```

### Default:
```
AuthLDAPSubGroupClass
groupOfNames
groupOfUniqueNames
```

### Context:
directory, .htaccess

### Override:
AuthConfig

### Status:
Extension

### Module:
mod_authnz_ldap

### Compatibility:
Available in version 2.3.0 and later

An LDAP group object may contain members that are users and members that are groups (called nested or sub groups). The `AuthLDAPSubGroupAttribute` directive identifies the labels of members that may be sub-groups of the current group (as opposed to user members). The `AuthLDAPSubGroupClass` directive specifies the LDAP objectClass values used in verifying that these potential sub-groups are in fact group objects. Verified sub-groups can then be searched for more user or sub-group members. Multiple attributes can be used by specifying this directive multiple times. If not specified, then `mod_authnz_ldap` uses the `groupOfNames` and `groupOfUniqueNames` values.
**Description:** URL specifying the LDAP search parameters

**Syntax:**

```
AuthLDAPUrl url
[NONE|SSL|TLS|STARTTLS]
```

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_authnz_ldap

An RFC 2255 URL which specifies the LDAP search parameters to use. The syntax of the URL is

```
```

If you want to specify more than one LDAP URL that Apache should try in turn, the syntax is:

```
AuthLDAPUrl "ldap://ldap1.example.com ldap2
```

**Caveat:** If you specify multiple servers, you need to enclose the entire URL string in quotes; otherwise you will get an error: "AuthLDAPURL takes one argument, URL to define LDAP connection." You can of course use search parameters on each of these.

**ldap**

For regular ldap, use the string ldap. For secure LDAP, use ldaps instead. Secure LDAP is only available if Apache was linked to an LDAP library with SSL support.

**host:port**

The name/port of the ldap server (defaults to localhost:389 for ldap, and localhost:636 for...
ldaps). To specify multiple, redundant LDAP servers, just list all servers, separated by spaces. **mod_authnz_ldap** will try connecting to each server in turn, until it makes a successful connection. If multiple ldap servers are specified, then entire LDAP URL must be encapsulated in double quotes.

Once a connection has been made to a server, that connection remains active for the life of the **httpd** process, or until the LDAP server goes down.

If the LDAP server goes down and breaks an existing connection, **mod_authnz_ldap** will attempt to re-connect, starting with the primary server, and trying each redundant server in turn. Note that this is different than a true round-robin search.

**basedn**
The DN of the branch of the directory where all searches should start from. At the very least, this must be the top of your directory tree, but could also specify a subtree in the directory.

**attribute**
The attribute to search for. Although RFC 2255 allows a comma-separated list of attributes, only the first attribute will be used, no matter how many are provided. If no attributes are provided, the default is to use u.id. It’s a good idea to choose an attribute that will be unique across all entries in the subtree you will be using. All attributes listed will be put into the environment with an AUTHENTICATE_ prefix for use by other modules.

**scope**
The scope of the search. Can be either one or sub. Note that a scope of base is also supported by RFC 2255, but is not supported by this module. If the scope is not provided, or if...
base scope is specified, the default is to use a scope of sub.

**filter**
A valid LDAP search filter. If not provided, defaults to (objectClass=*)
which will search for all objects in the tree. Filters are limited to approximately 8000 characters (the definition of MAX_STRING_LEN in the Apache source code). This should be more than sufficient for any application. In 2.4.10 and later, the keyword none disables the use of a filter; this is required by some primitive LDAP servers.

When doing searches, the attribute, filter and username passed by the HTTP client are combined to create a search filter that looks like (&(filter)(attribute=username)).

For example, consider an URL of ldap://ldap.example.com/o=Example?cn?sub?(posixid=*). When a client attempts to connect using a username of Babs Jenson, the resulting search filter will be (&(posixid=*)(cn=Babs Jenson)).

An optional parameter can be added to allow the LDAP Url to override the connection type. This parameter can be one of the following:

**NONE**
Establish an unsecure connection on the default LDAP port. This is the same as ldap:// on port 389.

**SSL**
Establish a secure connection on the default secure LDAP port. This is the same as ldaps://

**TLS | STARTTLS**
Establish an upgraded secure connection on the default LDAP port. This connection will be initiated on port 389 by
default and then upgraded to a secure connection on the same port.

See above for examples of AuthLDAPUrl URLs.
### Apache Module mod_authz_core

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Core Authorization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>authz_core_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_authz_core.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache HTTPD 2.3 and later</td>
</tr>
</tbody>
</table>

### Summary

This module provides core authorization capabilities so that authenticated users can be allowed or denied access to portions of the web site. mod_authz_core provides the functionality to register various authorization providers. It is usually used in conjunction with an authentication provider module such as mod_authn_file and an authorization module such as mod_authz_user. It also allows for advanced logic to be applied to the authorization processing.
Extended authorization providers can be created within the configuration file and assigned an alias name. The alias providers can then be referenced through the `Require` directive in the same way as a base authorization provider. Besides the ability to create and alias an extended provider, it also allows the same extended authorization provider to be referenced by multiple locations.

**Example**

The example below creates two different ldap authorization provider aliases based on the ldap-group authorization provider. This example allows a single authorization location to check group membership within multiple ldap hosts:

```xml
<AuthzProviderAlias ldap-group ldap-group-.alias1>
    AuthLDAPBindDN cn=youruser,o=ctx
    AuthLDAPBindPassword yourpassword
    AuthLDAPURL ldap://ldap.host/o=ctx
</AuthzProviderAlias>

<AuthzProviderAlias ldap-group ldap-group-alias2>
    AuthLDAPBindDN cn=yourotheruser,o=dev
    AuthLDAPBindPassword yourotherpassword
    AuthLDAPURL ldap://other.ldap.host/o=dev
</AuthzProviderAlias>

Alias "/secure" "/webpages/secure"
<Directory "/webpages/secure">
    Require all granted

    AuthBasicProvider file

    AuthType Basic
    AuthName LDAP.Protected.Place
</Directory>
```
# implied OR operation
Require ldap-group-alias1
Require ldap-group-alias2
</Directory>
The authorization container directives `<RequireAll>`, `<RequireAny>` and `<RequireNone>` may be combined with each other and with the `Require` directive to express complex authorization logic.

The example below expresses the following authorization logic. In order to access the resource, the user must either be the superadmin user, or belong to both the admins group and the Administrators LDAP group and either belong to the sales group or have the LDAP dept attribute sales. Furthermore, in order to access the resource, the user must not belong to either the temps group or the LDAP group Temporary Employees.

```
<Directory "/www/mydocs">
  <RequireAll>
    <RequireAny>
      Require user superadmin
      <RequireAll>
        Require group admins
        Require ldap-group cn=Administrators,o=Airius
      </RequireAll>
    </RequireAny>
  </RequireAll>
  <RequireAny>
    Require group temps
    Require ldap-group cn=Temporary Employees
  </RequireNone>
  </RequireAll>
</Directory>
```
**mod_authz_core** provides some generic authorization providers which can be used with the **Require** directive.

### Require env

The env provider allows access to the server to be controlled based on the existence of an *environment variable*. When **Require env env-variable** is specified, then the request is allowed access if the environment variable *env-variable* exists. The server provides the ability to set environment variables in a flexible way based on characteristics of the client request using the directives provided by **mod_setenvif**. Therefore, this directive can be used to allow access based on such factors as the clients User-Agent (browser type), Referer, or other HTTP request header fields.

```plaintext
SetEnvIf User-Agent ^KnockKnock/2\0 let_me_in
<Directory "/docroot">
  Require env let_me_in
</Directory>
```

In this case, browsers with a user-agent string beginning with KnockKnock/2.0 will be allowed access, and all others will be denied.

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.
**Require all**
The all provider mimics the functionality that was previously provided by the 'Allow from all' and 'Deny from all' directives. This provider can take one of two arguments which are 'granted' or 'denied'. The following examples will grant or deny access to all requests.

- Require all granted
- Require all denied

**Require method**
The method provider allows using the HTTP method in authorization decisions. The GET and HEAD methods are treated as equivalent. The TRACE method is not available to this provider, use TraceEnable instead.

The following example will only allow GET, HEAD, POST, and OPTIONS requests:

- Require method GET POST OPTIONS

The following example will allow GET, HEAD, POST, and OPTIONS requests without authentication, and require a valid user for all other methods:

```
<RequireAny>
  Require method GET POST OPTIONS
  Require valid-user
</RequireAny>
```

**Require expr**
The `expr` provider allows basing authorization decisions on arbitrary expressions.

The syntax is described in the [ap_expr documentation](#).

Normally, the expression is evaluated before authentication. However, if the expression returns false and references the variable `%{REMOTE_USER}`, authentication will be performed and the expression will be re-evaluated.
### Description
Controls the manner in which each configuration section's authorization logic is combined with that of preceding configuration sections.

### Syntax
AuthMerging Off | And | Or

### Default
AuthMerging Off

### Context
directory, .htaccess

### Override
AuthConfig

### Status
Base

### Module
mod_authz_core

When authorization is enabled, it is normally inherited by each subsequent configuration section, unless a different set of authorization directives is specified. This is the default action, which corresponds to an explicit setting of AuthMerging Off.

However, there may be circumstances in which it is desirable for a configuration section's authorization to be combined with that of its predecessor while configuration sections are being merged. Two options are available for this case, And and Or.

When a configuration section contains AuthMerging And or AuthMerging Or, its authorization logic is combined with that of the nearest predecessor (according to the overall order of configuration sections) which also contains authorization logic as if the two sections were jointly contained within a `<RequireAll>` or `<RequireAny>` directive, respectively.

The setting of AuthMerging is not inherited outside of the configuration section in which it appears. In the following example, only users belonging to group alpha may access /www/docs. Users belonging to either groups alpha or beta may access /www/docs/ab. However, the default Off setting
of AuthMerging applies to the <Directory> configuration section for /www/docs/ab/gamma, so that section's authorization directives override those of the preceding sections. Thus only users belong to the group gamma may access /www/docs/ab/gamma.

```xml
<Directory "/www/docs">
  AuthType Basic
  AuthName Documents
  AuthBasicProvider file
  AuthUserFile "/usr/local/apache/passwd/
  Require group alpha
</Directory>

<Directory "/www/docs/ab">
  AuthMerging Or
  Require group beta
</Directory>

<Directory "/www/docs/ab/gamma">
  Require group gamma
</Directory>
```
**Description:** Enclose a group of directives that represent an extension of a base authorization provider and referenced by the specified alias

**Syntax:**
```
<AuthzProviderAlias baseProvider Alias Require-Parameters> ... </AuthzProviderAlias>
```

**Context:** server config

**Status:** Base

**Module:** mod_authz_core

<AuthzProviderAlias> and </AuthzProviderAlias> are used to enclose a group of authorization directives that can be referenced by the alias name using the directive Require.
| **Description:** | Send '403 FORBIDDEN' instead of '401 UNAUTHORIZED' if authentication succeeds but authorization fails |
| **Syntax:** | AuthzSendForbiddenOnFailure On|Off |
| **Default:** | AuthzSendForbiddenOnFailure Off |
| **Context:** | directory, .htaccess |
| **Status:** | Base |
| **Module:** | mod_authz_core |
| **Compatibility:** | Available in Apache HTTPD 2.3.11 and later |

If authentication succeeds but authorization fails, Apache HTTPD will respond with an HTTP response code of '401 UNAUTHORIZED' by default. This usually causes browsers to display the password dialogue to the user again, which is not wanted in all situations. AuthzSendForbiddenOnFailure allows to change the response code to '403 FORBIDDEN'.

**Security Warning**

Modifying the response in case of missing authorization weakens the security of the password, because it reveals to a possible attacker, that his guessed password was right.
**Description:** Tests whether an authenticated user is authorized by an authorization provider.

**Syntax:**
```
Require [not] entity-name [entity-name] ...
```

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_authz_core

This directive tests whether an authenticated user is authorized according to a particular authorization provider and the specified restrictions. mod_authz_core provides the following generic authorization providers:

**Require all granted**
Access is allowed unconditionally.

**Require all denied**
Access is denied unconditionally.

**Require env env-var [env-var] ...**
Access is allowed only if one of the given environment variables is set.

Access is allowed only for the given HTTP methods.

**Require expr expression**
Access is allowed if expression evaluates to true.

Some of the allowed syntaxes provided by mod_authz_user, mod_authz_host, and mod_authz_groupfile are:

**Require user userid [userid] ...**
Only the named users can access the resource.
**Require group** *group-name*  *[group-name]* ...

Only users in the named groups can access the resource.

**Require valid-user**

All valid users can access the resource.

**Require ip 10 172.20 192.168.2**

Clients in the specified IP address ranges can access the resource.

Other authorization modules that implement require options include *mod_authnz_ldap*, *mod_authz_dbm*, *mod_authz_dbd*, *mod_authz_owner* and *mod_ssl*.

In most cases, for a complete authentication and authorization configuration, **Require** must be accompanied by **AuthName**, **AuthType** and **AuthBasicProvider** or **AuthDigestProvider** directives, and directives such as **AuthUserFile** and **AuthGroupFile** (to define users and groups) in order to work correctly. Example:

```
AuthType Basic
AuthName "Restricted Resource"
AuthBasicProvider file
AuthUserFile "/web/users"
AuthGroupFile "/web/groups"
Require group admin
```

Access controls which are applied in this way are effective for all methods. **This is what is normally desired.** If you wish to apply access controls only to specific methods, while leaving other methods unprotected, then place the **Require** statement into a **<Limit>** section.

The result of the **Require** directive may be negated through the
use of the not option. As with the other negated authorization directive <RequireNone>, when the Require directive is negated it can only fail or return a neutral result, and therefore may never independently authorize a request.

In the following example, all users in the alpha and beta groups are authorized, except for those who are also in the reject group.

```
<Directory "/www/docs">
  <RequireAll>
    Require group alpha beta
    Require not group reject
  </RequireAll>
</Directory>
```

When multiple Require directives are used in a single configuration section and are not contained in another authorization directive like <RequireAll>, they are implicitly contained within a <RequireAny> directive. Thus the first one to authorize a user authorizes the entire request, and subsequent Require directives are ignored.

**Security Warning**

Exercise caution when setting authorization directives in Location sections that overlap with content served out of the filesystem. By default, these configuration sections overwrite authorization configuration in Directory, and Files sections.

The AuthMerging directive can be used to control how authorization configuration sections are merged.

See also
- **Access control howto**
- **Authorization Containers**
- **mod_authn_core**
- **mod_authz_host**
**Description:** Enclose a group of authorization directives of which none must fail and at least one must succeed for the enclosing directive to succeed.

**Syntax:** `<RequireAll> ... </RequireAll>`

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_authz_core

`<RequireAll>` and `</RequireAll>` are used to enclose a group of authorization directives of which none must fail and at least one must succeed in order for the `<RequireAll>` directive to succeed.

If none of the directives contained within the `<RequireAll>` directive fails, and at least one succeeds, then the `<RequireAll>` directive succeeds. If none succeed and none fail, then it returns a neutral result. In all other cases, it fails.

**See also**

- [Authorization Containers](#)
- [Authentication, Authorization, and Access Control](#)
**Description:** Enclose a group of authorization directives of which one must succeed for the enclosing directive to succeed.

**Syntax:** `<RequireAny> ... </RequireAny>`

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_authz_core

`<RequireAny>` and `</RequireAny>` are used to enclose a group of authorization directives of which one must succeed in order for the `<RequireAny>` directive to succeed.

If one or more of the directives contained within the `<RequireAny>` directive succeed, then the `<RequireAny>` directive succeeds. If none succeed and none fail, then it returns a neutral result. In all other cases, it fails.

Because negated authorization directives are unable to return a successful result, they can not significantly influence the result of a `<RequireAny>` directive. (At most they could cause the directive to fail in the case where they failed and all other directives returned a neutral value.) Therefore negated authorization directives are not permitted within a `<RequireAny>` directive.

**See also**

- [Authorization Containers](#)
- [Authentication, Authorization, and Access Control](#)
**Description:** Enclose a group of authorization directives of which none must succeed for the enclosing directive to not fail.

**Syntax:** `<RequireNone> ... </RequireNone>`

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Base

**Module:** mod_authz_core

`<RequireNone>` and `</RequireNone>` are used to enclose a group of authorization directives of which none must succeed in order for the `<RequireNone>` directive to not fail.

If one or more of the directives contained within the `<RequireNone>` directive succeed, then the `<RequireNone>` directive fails. In all other cases, it returns a neutral result. Thus as with the other negated authorization directive `Require not`, it can never independently authorize a request because it can never return a successful result. It can be used, however, to restrict the set of users who are authorized to access a resource.

Because negated authorization directives are unable to return a successful result, they can not significantly influence the result of a `<RequireNone>` directive. Therefore negated authorization directives are not permitted within a `<RequireNone>` directive.

**See also**

- [Authorization Containers](#)
- [Authentication, Authorization, and Access Control](#)
Apache Module mod_authz_dbd

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Group Authorization and Login using SQL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>authz_dbd_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_authz_dbd.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.4 and later</td>
</tr>
</tbody>
</table>

**Summary**

This module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the web site by group membership. Similar functionality is provided by `mod_authz_groupfile` and `mod_authz_dbm`, with the exception that this module queries a SQL database to determine whether a user is a member of a group.

This module can also provide database-backed user login/logout capabilities. These are likely to be of most value when used in conjunction with `mod_authn_dbd`.

This module relies on `mod_dbd` to specify the backend database driver and connection parameters, and manage the database connections.

**Bugfix checklist**

- [httpd changelog](http://httpd.apache.org/docs/2.4/changelog.html)
- [Known issues](http://httpd.apache.org/known-issues.html)
See also

Require
AuthDBDUserPWQuery
DBDriver
DBDParams
Apache'sRequire directives are used during the authorization phase to ensure that a user is allowed to access a resource. mod_authz_dbd extends the authorization types with dbd-group, dbd-login and dbd-logout.

Since v2.4.8, expressions are supported within the DBD require directives.

**Require dbd-group**
This directive specifies group membership that is required for the user to gain access.

```
Require dbd-group team
AuthzDBDQuery "SELECT group FROM authz WHERE...
```

**Require dbd-login**
This directive specifies a query to be run indicating the user has logged in.

```
Require dbd-login
AuthzDBDQuery "UPDATE authn SET login = 'true'
```

**Require dbd-logout**
This directive specifies a query to be run indicating the user has logged out.

```
Require dbd-logout
AuthzDBDQuery "UPDATE authn SET login = 'false'
```
In addition to the standard authorization function of checking group membership, this module can also provide server-side user session management via database-backed login/logout capabilities. Specifically, it can update a user's session status in the database whenever the user visits designated URLs (subject of course to users supplying the necessary credentials).

This works by defining two special **Require** types: **Require dbd-login** and **Require dbd-logout**. For usage details, see the configuration example below.
Some administrators may wish to implement client-side session management that works in concert with the server-side login/logout capabilities offered by this module, for example, by setting or unsetting an HTTP cookie or other such token when a user logs in or out.

To support such integration, mod_authz_dbd exports an optional hook that will be run whenever a user's status is updated in the database. Other session management modules can then use the hook to implement functions that start and end client-side sessions.
# mod_dbd configuration
DBDriver pgsql
DBDParams "dbname=apacheauth user=apache pass=xxxxx"

DBDMin 4
DBDKeep 8
DBDMax 20
DBDExptime 300

<Directory "/usr/www/my.site/team-private/">
  # mod_authn_core and mod_auth_basic configuration
  # for mod_authn_dbd
  AuthType Basic
  AuthName Team
  AuthBasicProvider dbd

  # mod_authn_dbd SQL query to authenticate
  AuthDBDUserPWQuery "SELECT password FROM authn WHERE user = %s AND login = 'true'"

  # mod_authz_core configuration for mod_authz_dbd
  Require dbd-group team

  # mod_authz_dbd configuration
  AuthzDBDQuery "SELECT group FROM authz WHERE user = %s"

  # when a user fails to be authenticated or
  # invite them to login; this page should provide
  # to /team-private/login.html
  ErrorDocument 401 "/login-info.html"

<Files "login.html">
  # don't require user to already be logged
  AuthDBDUserPWQuery "SELECT password FROM authn WHERE user = %s"
# dbd-login action executes a statement
Require dbd-login
AuthzDBDQuery "UPDATE authn SET login = 'true' WHERE user = %s"

# return user to referring page (if any)
# successful login
AuthzDBDLoginToReferer On
</Files>

<Files "logout.html">
# dbd-logou action executes a statement
Require dbd-logou
AuthzDBDQuery "UPDATE authn SET login = 'false' WHERE user = %s"
</Files>
</Directory>
**Description:** Determines whether to redirect the Client to the Referring page on successful login or logout if a Referer request header is present

**Syntax:** AuthzDBDLoginToReferer On|Off

**Default:** AuthzDBDLoginToReferer Off

**Context:** directory

**Status:** Extension

**Module:** mod_authz_dbd

In conjunction with Require dbd-login or Require dbd-logout, this provides the option to redirect the client back to the Referring page (the URL in the Referer HTTP request header, if present). When there is no Referer header, AuthzDBDLoginToReferer On will be ignored.
**Description:** Specify the SQL Query for the required operation

**Syntax:** AuthzDBDQuery query

**Context:** directory

**Status:** Extension

**Module:** mod_authz_dbd

The AuthzDBDQuery specifies an SQL query to run. The purpose of the query depends on the Require directive in effect.

- When used with a Require dbd-group directive, it specifies a query to look up groups for the current user. This is the standard functionality of other authorization modules such as mod_authz_groupfile and mod_authz_dbm. The first column value of each row returned by the query statement should be a string containing a group name. Zero, one, or more rows may be returned.

```
Require dbd-group
AuthzDBDQuery "SELECT group FROM groups WHERE user = %s"
```

- When used with a Require dbd-login or Require dbd-logout directive, it will never deny access, but will instead execute a SQL statement designed to log the user in or out. The user must already be authenticated with mod_authn_dbd.

```
Require dbd-login
AuthzDBDQuery "UPDATE authn SET login = %s"
```

In all cases, the user's ID will be passed as a single string...
parameter when the SQL query is executed. It may be referenced within the query statement using a %s format specifier.
**Description:** Specify a query to look up a login page for the user

**Syntax:**

```
AuthzDBDRedirectQuery query
```

**Context:** directory

**Status:** Extension

**Module:** mod_authz_dbd

Specifies an optional SQL query to use after successful login (or logout) to redirect the user to a URL, which may be specific to the user. The user's ID will be passed as a single string parameter when the SQL query is executed. It may be referenced within the query statement using a `%s` format specifier.

```
AuthzDBDRedirectQuery "SELECT userpage FROM userpages WHERE user = %s"
```

The first column value of the first row returned by the query statement should be a string containing a URL to which to redirect the client. Subsequent rows will be ignored. If no rows are returned, the client will not be redirected.

**Note** that `AuthzDBDLoginToReferer` takes precedence if both are set.

---

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](https://www.apache.org/licenses/LICENSE-2.0).
### Apache Module mod_authz_dbm

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Group authorization using DBM files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>authz_dbm_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_authz_dbm.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.1 and later</td>
</tr>
</tbody>
</table>

**Summary**

This module provides authorization capabilities so that authenticated users can be allowed or denied access to portions of the web site by group membership. Similar functionality is provided by `mod_authz_groupfile`.

**Bugfix checklist**

- [httpd changelog](httpd changelog)
- [Known issues](Known issues)
- [Report a bug](Report a bug)

**See also**

- [Require](Require)
Apache's Require directives are used during the authorization phase to ensure that a user is allowed to access a resource. mod_authz_dbm extends the authorization types with dbm-group.

Since v2.4.8, expressions are supported within the DBM require directives.

**Require dbm-group**

This directive specifies group membership that is required for the user to gain access.

```
Require dbm-group admin
```

**Require dbm-file-group**

When this directive is specified, the user must be a member of the group assigned to the file being accessed.

```
Require dbm-file-group
```
Note that using mod_authz_dbm requires you to require dbm-group instead of group:

```xml
<Directory "/foo/bar">
  AuthType Basic
  AuthName "Secure Area"
  AuthBasicProvider dbm
  AuthDBMUserFile "site/data/users"
  AuthDBMGroupFile "site/data/users"
  Require dbm-group admin
</Directory>
```
**Description:** Sets the name of the database file containing the list of user groups for authorization

**Syntax:**

```
AuthDBMGroupFile file-path
```

**Context:**

directory, .htaccess

**Override:**

AuthConfig

**Status:**

Extension

**Module:**

mod_authz_dbm

The **AuthDBMGroupFile** directive sets the name of a DBM file containing the list of user groups for user authorization. *File-path* is the absolute path to the group file.

The group file is keyed on the username. The value for a user is a comma-separated list of the groups to which the users belongs. There must be no whitespace within the value, and it must never contain any colons.

**Security**

Make sure that the **AuthDBMGroupFile** is stored outside the document tree of the web-server. Do **not** put it in the directory that it protects. Otherwise, clients will be able to download the **AuthDBMGroupFile** unless otherwise protected.

Combining Group and Password DBM files: In some cases it is easier to manage a single database which contains both the password and group details for each user. This simplifies any support programs that need to be written: they now only have to deal with writing to and locking a single DBM file. This can be accomplished by first setting the group and password files to point to the same DBM:

```
AuthDBMGroupFile "/www/userbase"
```
The key for the single DBM is the username. The value consists of

```
Encrypted Password : List of Groups [ : (ignored) ]
```

The password section contains the encrypted password as before. This is followed by a colon and the comma separated list of groups. Other data may optionally be left in the DBM file after another colon; it is ignored by the authorization module. This is what www.telescope.org uses for its combined password and group database.
### Description
Sets the type of database file that is used to store the list of user groups.

### Syntax
```
AuthzDBMType
default|SDBM|GDBM|NDBM|DB
```

### Default
```
AuthzDBMType default
```

### Context
directory, .htaccess

### Override
AuthConfig

### Status
Extension

### Module
```
mod_authz_dbm
```

Sets the type of database file that is used to store the list of user groups. The default database type is determined at compile time. The availability of other types of database files also depends on compile-time settings.

It is crucial that whatever program you use to create your group files is configured to use the same type of database.
Apache mod_authz_groupfile

Bugfix checklist

httpd changelog
Known issues
Report a bug

Require
AuthGroupFile file-path

AuthGroupFile

mygroup: bob joe anne

AuthDBMGroupFile
Apache Module mod_authz_host

**Description:** Group authorizations based on host (name or IP address)

**Status:** Base

**Module Identifier:** authz_host_module

**Source File:** mod_authz_host.c

**Compatibility:** The forward-dns provider was added in 2.4.19

### Summary

The authorization providers implemented by mod_authz_host are registered using the `Require` directive. The directive can be referenced within a `<Directory>`, `<Files>`, or `<Location>` section as well as `.htaccess` files to control access to particular parts of the server. Access can be controlled based on the client hostname or IP address.

In general, access restriction directives apply to all access methods (GET, PUT, POST, etc). This is the desired behavior in most cases. However, it is possible to restrict some methods, while leaving other methods unrestricted, by enclosing the directives in a `<Limit>` section.

### Bugfix checklist

- [httpd changelog](http://httpd.apache.org/docs/latest/changelog.html)
- [Known issues](http://httpd.apache.org/reqdocs/known-issues.html)
Report a bug

See also

Authentication, Authorization, and Access Control
Require
Apache's `Require` directive is used during the authorization phase to ensure that a user is allowed or denied access to a resource. `mod_authz_host` extends the authorization types with `ip`, `host`, `forward-dns` and `local`. Other authorization types may also be used but may require that additional authorization modules be loaded.

These authorization providers affect which hosts can access an area of the server. Access can be controlled by hostname, IP Address, or IP Address range.

Since v2.4.8, expressions are supported within the host require directives.

**Require ip**

The `ip` provider allows access to the server to be controlled based on the IP address of the remote client. When `Require ip ip-address` is specified, then the request is allowed access if the IP address matches.

A full IP address:

```
Require ip 10.1.2.3
Require ip 192.168.1.104 192.168.1.205
```

An IP address of a host allowed access

A partial IP address:

```
Require ip 10.1
Require ip 10 172.20 192.168.2
```

The first 1 to 3 bytes of an IP address, for subnet restriction.
A network/netmask pair:

```plaintext
Require ip 10.1.0.0/255.255.0.0
```

A network a.b.c.d, and a netmask w.x.y.z. For more fine-grained subnet restriction.

A network/nnn CIDR specification:

```plaintext
Require ip 10.1.0.0/16
```

Similar to the previous case, except the netmask consists of nnn high-order 1 bits.

Note that the last three examples above match exactly the same set of hosts.

IPv6 addresses and IPv6 subnets can be specified as shown below:

```plaintext
Require ip 2001:db8::a00:20ff:fea7:ccea
Require ip 2001:db8:1:1::a
Require ip 2001:db8:2:1::/64
Require ip 2001:db8:3::/48
```

Note: As the IP addresses are parsed on startup, expressions are not evaluated at request time.

**Require host**

The host provider allows access to the server to be controlled based on the host name of the remote client. When `Require host host-name` is specified, then the request is allowed access if the host name matches.
Hosts whose names match, or end in, this string are allowed access. Only complete components are matched, so the above example will match foo.example.org but it will not match fooexample.org. This configuration will cause Apache to perform a double reverse DNS lookup on the client IP address, regardless of the setting of the HostnameLookups directive. It will do a reverse DNS lookup on the IP address to find the associated hostname, and then do a forward lookup on the hostname to assure that it matches the original IP address. Only if the forward and reverse DNS are consistent and the hostname matches will access be allowed.

**Require forward-dns**

The forward-dns provider allows access to the server to be controlled based on simple host names. When Require forward-dns *host-name* is specified, all IP addresses corresponding to *host-name* are allowed access.

In contrast to the host provider, this provider does not rely on reverse DNS lookups: it simply queries the DNS for the host name and allows a client if its IP matches. As a consequence, it will only work with host names, not domain names. However, as the reverse DNS is not used, it will work with clients which use a dynamic DNS service.

**Require forward-dns bla.example.org**
bla.example.org will be granted access.

The forward-dns provider was added in 2.4.19.

**Require local**

The `local` provider allows access to the server if any of the following conditions is true:

- the client address matches 127.0.0.0/8
- the client address is ::1
- both the client and the server address of the connection are the same

This allows a convenient way to match connections that originate from the local host:

```
Require local
```

**Security Note**

If you are proxying content to your server, you need to be aware that the client address will be the address of your proxy server, not the address of the client, and so using the `Require` directive in this context may not do what you mean. See `mod_remoteip` for one possible solution to this problem.
Apache mod_authz_owner

:: Extension
:: authz_owner_module
:: mod_authz_owner.c
:: Apache 2.1

HTTP ID (ID)

mod_auth_digest
file-owner file-group:

file-owner

file-group

mod_authz_groupfile
() accounts accounts

() mod_authz_groupfile

"MultiViews"
Bugfix checklist

httpd changelog
Known issues
Report a bug

Require
Require file-owner

Apache

AuthDBMUserFile
/home/smith/public_html/private

jones

<Directory /home/*/public_html/private>
  AuthType Basic
  AuthName MyPrivateFiles
  AuthBasicProvider dbm
  AuthDBMUserFile /usr/local/apache2/etc/.htdbm-all
  Require file-owner
</Directory>

Require file-group

AuthDBMGroupFile
foo project-foo

<Directory /home/*/public_html/project-foo>
  AuthType Basic
  AuthName "Project Foo Files"
  AuthBasicProvider dbm

  # combined user/group database
  AuthDBMUserFile /usr/local/apache2/etc/.htdbm-all
  AuthDBMGroupFile /usr/local/apache2/etc/.htdbm-all

  Satisfy All
  Require file-group
</Directory>
Apache mod_authz_user

valid-user

Bugfix checklist

httpd changelog
Known issues
Report a bug

Require
Apache mod_autoindex

<table>
<thead>
<tr>
<th>Unix</th>
<th>ls</th>
<th>Win32</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>dir</td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td></td>
<td></td>
</tr>
<tr>
<td>autoindex_module</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mod_autoindex.c</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- index.html
- DirectoryIndex
- AddIconByType

Options +Indexes

FancyIndexing
IndexOptions
SuppressColumnSorting

"Size"
Apache 2.0.23

- C=N
- C=M
- C=S
- C=D
- O=A
- O=D
- F=0  (FancyIndex)
- F=1 FancyIndex
- F=2 HTML FancyIndex
- V=0
- V=1
- P=pattern pattern

"P(P)" IndexIgnore mod_autoindex ()

HEADER.html

<form action="" method="get">
  Show me a <select name="F">
    <option value="0">Plain list</option>
    <option value="1" selected="selected">Fancy list</option>
    <option value="2">Table list</option>
  </select>
  Sorted by <select name="C">
    <option value="N" selected="selected">Name</option>
    <option value="M">Date Modified</option>
    <option value="S">Size</option>
  </select>
</form>
<option value="D"> Description</option>
</select>
<select name="O">
    <option value="A" selected="selected"> Ascending</option>
    <option value="D"> Descending</option>
</select>
<select name="V">
    <option value="0" selected="selected"> in Normal order</option>
    <option value="1"> in Version order</option>
</select>
Matching <input type="text" name="P" value="*" />
<input type="submit" name="X" value="Go" />
</form>
AddAlt string file [file] ...
AddAlt ,., .htaccess
AddAlt Indexes
AddAlt Base
AddAlt mod_autoindex

AddAlt FancyIndexing
        (" ")

AddAlt "PDF file" *.pdf
AddAlt Compressed *.gz *.zip *Z
AddAltByEncoding string MIME-encoding [MIME-encoding] ...

.,.,.htaccess
Indexes
Base
mod_autoindex

AddAltByEncoding FancyIndexing
encoding x-compress string ( " ')

AddAltByEncoding gzip x-gzip
AddAltByType string MIME-type [MIME-type] ...

.,.,.htaccess

Indexes

Base

mod_autoindex

AddAltByType FancyIndexing
text/html string ( " " )

AddAltByType 'plain text' text/plain
AddDescription *string file* [file] ...

AddDescription ...,.htaccess

Indexes

Base

mod_autoindex

FancyIndexing

AddDescription "The planet Mars" /web/pics/mars.gif

23

IndexOptions SuppressSize 7

IndexOptions SuppressLastModified 19

AddDescription HTML
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AddIcon icon name [name]</td>
<td>Adds an icon to files</td>
</tr>
<tr>
<td>.htaccess</td>
<td>Adds the .htaccess file to directories</td>
</tr>
<tr>
<td>FancyIndexing name</td>
<td>Selects how directories are indexed</td>
</tr>
<tr>
<td>(alttext, url) alttext</td>
<td>Specifies the alternate text and URL for directories</td>
</tr>
<tr>
<td>name ^^DIRECTORY^^</td>
<td>Specifies directory icons</td>
</tr>
<tr>
<td>^^BLANKICON^^ (</td>
<td>Specifies blank icon for directories</td>
</tr>
<tr>
<td>AddIcon (IMG,/icons/image.xbm) .gif .jpg .xbm</td>
<td>Adds icons to files</td>
</tr>
<tr>
<td>AddIcon /icons/dir.xbm ^^DIRECTORY^^</td>
<td>Adds directory icon</td>
</tr>
<tr>
<td>AddIcon /icons/backup.xbm *~</td>
<td>Adds backup icon</td>
</tr>
</tbody>
</table>

AddIcon AddIconByType
AddIconByEncoding icon MIME-encoding [MIME-encoding] ...

,, .htaccess
Indexes
 Base
mod_autoindex

FancyIndexing (alttext, url) alttext

MIME-encoding x-compress

AddIconByEncoding /icons/compress.xbm x-compress
AddIconByType icon MIME-type [MIME-type] ...

,,.htaccess

Indexes

Base

mod_autoindex

FancyIndexing

(alttex, url) alttext

MIME-type

AddIconByType (IMG,/icons/image.xbm) image/*
<table>
<thead>
<tr>
<th>Directory Options</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>DefaultIcon url-path</code></td>
</tr>
<tr>
<td><code>htaccess</code></td>
</tr>
<tr>
<td>Indexes</td>
</tr>
<tr>
<td>Base</td>
</tr>
<tr>
<td><code>mod_autoindex</code></td>
</tr>
</tbody>
</table>

FancyIndexing

DefaultIcon /icon/unknown.xbm
<table>
<thead>
<tr>
<th>HeaderName</th>
<th>filename</th>
<th>URI</th>
<th>DocumentRoot</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HeaderName</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>filename .htaccess</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Indexes</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Base</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>mod_autoindex</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HeaderName</th>
<th>ReadmeName</th>
<th>filename</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HEADER.html</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>filename</th>
<th>&quot;</th>
<th>text/*&quot;</th>
<th>(text/html, text/</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CGI</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Options</th>
<th>MultiViews</th>
<th>file</th>
</tr>
</thead>
<tbody>
<tr>
<td>text/html</td>
<td>options Includes</td>
<td>Include</td>
</tr>
<tr>
<td></td>
<td>(mod_include)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HeaderName</th>
<th>HTML (&lt;html&gt;, &lt;head&gt;,</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IndexOptions +SuppressHTMLPreamble</td>
</tr>
</tbody>
</table>
HEAD
IndexHeadInsert "markup ..."
.,.,.htaccess
Indexes
Base
mod_autoindex

Example
IndexHeadInsert "<link rel="sitemap" href="/sitemap.html"">"
IndexIgnore
()

IndexIgnore README .htaccess *.bak *~
Empties the list of files to hide when listing a directory

IndexIgnoreReset  ON|OFF

., ., .htaccess

Indexes

Base

mod_autoindex

2.3.10 and later
IndexOptions

Option:

Charset=character-set (Apache 2.0.61)

Example:
IndexOptions Charset=UTF-8

Type=MIME content-type (Apache 2.0.61)

Example:
IndexOptions Type=text/plain

DescriptionWidth=[n | *] (2.0.23)

Example:
IndexOptions DescriptionWidth=* AddDescription

FancyIndexing

FoldersFirst (2.0.23)
Zed Beta Gamma

**HTMLTable** (Apache 2.0.23)
FancyIndexing

**IconsAreLinks**
FancyIndexing

**IconHeight[=pixels]**
IconWidth
Apache

**IconWidth[=pixels]**
IconHeight
Apache

**IgnoreCase**

gamma )

**IgnoreClient**
mod_autoindex

**SuppressColumnSorting**

**NameWidth=[n | *]**
NameWidth

-NameWidth ()

NameWidth=n n
NameWidth=*

**ScanHTMLTitles**
FancyIndexing HTML

httpd title CPU (C

**ShowForbidden**
Apache HTTP_UNAUTHORIZED HTTP_FORBIDDEN
SuppressColumnSorting
  Apache FancyIndexing
2.0.23
IgnoreClient

SuppressDescription
  FancyIndexing
  AddDescription

SuppressHTMLPreamble
  HeaderName HTML
  SuppressHTMLPreamble

SuppressIcon (Apache 2.0.23 )
  FancyIndexing SuppressIcon
  HTML 3.2 HTML 3.2 img hr (FancyIndexing )

SuppressLastModified
  FancyIndexing

SuppressRules (Apache 2.0.23 )
  ( hr ) SuppressIcon SuppressRules
  HTML 3.2 HTML 3.2 img hr |
  (FancyIndexing )

SuppressSize
  FancyIndexing

TrackModified (Apache 2.0.23 )
  HTTP Last-Modified ETag
  UNIX OS2 JFS Win32 NTFS
  Win32 FAT
  Last-Modified

VersionSort (Apache 2.0a3 )
  VersionSort
- foo-1.7
- foo-1.7.2
- foo-1.7.12
- foo-1.8.2
- foo-1.8.2a
- foo-1.12

0

- foo-1.001
- foo-1.002
- foo-1.030
- foo-1.04

**XHTML (Apache 2.0.49)**

- `mod_autoindex`
- HTML 3.2
- XHTML 1.0

**IndexOptions**

- Apache 1.3.3

  - IndexOptions

    <Directory /foo>
    
    IndexOptions HTMLTable
    IndexOptions SuppressColumnsorting
    
    </Directory>

  - IndexOptions HTMLTable SuppressColumnsorting

    - ('+' '-' )

    ' + ' - '  

    IndexOp
IndexOptions +ScanHTMLTitles -IconsAreLinks FancyIndexing
IndexOptions +SuppressSize

IndexOptions FancyIndexing +SuppressSize
FancyIndexing
IndexOrderDefault Ascending|Descending
Name|Date|Size|Description
IndexOrderDefault Ascending Name
,.,.htaccess
Indexes
Base
mod_autoindex

IndexOrderDefault  FancyIndexing
IndexOrderDefault
Name, Date, Size  Description 1

SuppressColumnSorting
IndexStyleSheet

CSS

IndexStyleSheet  url-path

,.,.htaccess

Indexes

Base

mod_autoindex

IndexStyleSheet  

"/css/style.css"
ReadmeName  
ReadmeName filename

.,., .htaccess

Indexes

Base

mod_autoindex

ReadmeName DocumentRoot

ReadmeName FOOTER.html

2

ReadmeName /include/FOOTER.html

HeaderName
Apache Module mod_brotli

<table>
<thead>
<tr>
<th>Description</th>
<th>Compress content via Brothli before it is delivered to the client</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Extension</td>
</tr>
<tr>
<td>Module Identifier</td>
<td>brotli_module</td>
</tr>
<tr>
<td>Source File</td>
<td>mod_brotli.c</td>
</tr>
<tr>
<td>Compatibility</td>
<td>Available in version 2.4.26 and later.</td>
</tr>
</tbody>
</table>

Summary

The mod_brotli module provides the BROTLI_COMPRESS output filter that allows output from your server to be compressed using the brotli compression format before being sent to the client over the network. This module uses the Brothli library found at https://github.com/google/brotli.

Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

Filters
Compression and TLS

Some web applications are vulnerable to an information disclosure attack when a TLS connection carries compressed data. For more information, review the details of the "BREACH" family of attacks.

This is a simple configuration that compresses common text-based content types.

Compress only a few types

```
AddOutputFilterByType BROTLI_COMPRESS text/html text/plain text/xml text/css text/javascript application/javascript
```
Compression and TLS

Some web applications are vulnerable to an information disclosure attack when a TLS connection carries compressed data. For more information, review the details of the "BREACH" family of attacks.

Output Compression

Compression is implemented by the BROTLI_COMPRESS filter. The following directive will enable compression for documents in the container where it is placed:

```
SetOutputFilter BROTLI_COMPRESS
SetEnvIfNoCase Request_URI \.(?:gif|jpe?g|png)$ no-brotli
```

If you want to restrict the compression to particular MIME types in general, you may use the AddOutputFilterByType directive. Here is an example of enabling compression only for the html files of the Apache documentation:

```
<Directory "/your-server-root/manual">
    AddOutputFilterByType BROTLI_COMPRESS text/html
</Directory>
```

Note

The BROTLI_COMPRESS filter is always inserted after RESOURCE filters like PHP or SSI. It never touches internal subrequests.
There is an environment variable no-brotli, set via `SetEnv`, which will disable brotli compression for a particular request, even if it is supported by the client.
The `mod_brotli` module sends a Vary: Accept-Encoding HTTP response header to alert proxies that a cached response should be sent only to clients that send the appropriate Accept-Encoding request header. This prevents compressed content from being sent to a client that will not understand it.

If you use some special exclusions dependent on, for example, the User-Agent header, you must manually configure an addition to the Vary header to alert proxies of the additional restrictions. For example, in a typical configuration where the addition of the BROTLI_COMPRESS filter depends on the User-Agent, you should add:

```
Header append Vary User-Agent
```

If your decision about compression depends on other information than request headers (e.g. HTTP version), you have to set the Vary header to the value *. This prevents compliant proxies from caching entirely.

```Example
Header set Vary *
```
Since mod_brotli re-compresses content each time a request is made, some performance benefit can be derived by pre-compressing the content and telling mod_brotli to serve them without re-compressing them. This may be accomplished using a configuration like the following:

```html
<IfModule mod_headers.c>

    # Serve brotli compressed CSS files if they exist and the client accepts brotli.
    RewriteCond %{HTTP:Accept-encoding} br
    RewriteCond %{REQUEST_FILENAME}.br -s
    RewriteRule ^\.(.*)\.css$ $1\.css\.br [QSA]

    # Serve brotli compressed JS files if they exist and the client accepts brotli.
    RewriteCond %{HTTP:Accept-encoding} br
    RewriteCond %{REQUEST_FILENAME}.br -s
    RewriteRule ^\.(.*)\.js$ $1\.js\.br [QSA]

    # Serve correct content types, and prevent double compression.
    RewriteRule \.(css|js)\.br$ - [T=text/css,E=no-brotli:1]

    <FilesMatch "\.(js\.br|css\.br)$">
        # Serve correct encoding type.
        Header append Content-Encoding br
    </FilesMatch>

    # Force proxies to cache brotli & non-brotli css/js files separately.
    Header append Vary Accept-Encoding

</IfModule>
```
**Description:** How the outgoing ETag header should be modified during compression

**Syntax:** BrotliAlterETag
AddSuffix|NoChange|Remove

**Default:** BrotliAlterETag AddSuffix

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_brotli

The **BrotliAlterETag** directive specifies how the ETag header should be altered when a response is compressed.

**AddSuffix**
Append the compression method onto the end of the ETag, causing compressed and uncompressed representations to have unique ETags. In another dynamic compression module, mod_deflate, this has been the default since 2.4.0. This setting prevents serving "HTTP Not Modified" (304) responses to conditional requests for compressed content.

**NoChange**
Don't change the ETag on a compressed response. In another dynamic compression module, mod_deflate, this has been the default prior to 2.4.0. This setting does not satisfy the HTTP/1.1 property that all representations of the same resource have unique ETags.

**Remove**
Remove the ETag header from compressed responses. This prevents some conditional requests from being possible, but avoids the shortcomings of the preceding options.
**Description:** Maximum input block size

**Syntax:** BrotliCompressionMaxInputBlock value

**Default:** (automatic)

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_brotli

The `BrotliCompressionMaxInputBlock` directive specifies the maximum input block size between 16 and 24, with the caveat that larger block sizes require more memory.
**Description:** Compression quality

**Syntax:** BrotliCompressionQuality value

**Default:** BrotliCompressionQuality 5

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_brotli

The `BrotliCompressionQuality` directive specifies the compression quality (a value between 0 and 11). Higher quality values result in better, but also slower compression.
The **BrotliCompressionWindow** directive specifies the brotli sliding compression window size (a value between 10 and 24). Larger window sizes can improve compression quality, but require more memory.
The BrotliFilterNote directive specifies that a note about compression ratios should be attached to the request. The name of the note is the value specified for the directive. You can use that note for statistical purposes by adding the value to your access log.

Example

BrotliFilterNote ratio

LogFormat "%r" %b (%{ratio}n) "%{User-agent}i" brotli
CustomLog "logs/brotli_log" brotli

If you want to extract more accurate values from your logs, you can use the type argument to specify the type of data left as a note for logging. type can be one of:

**Input**
- Store the byte count of the filter's input stream in the note.

**Output**
- Store the byte count of the filter's output stream in the note.

**Ratio**
- Store the compression ratio (output/input * 100) in the note. This is the default, if the type argument is omitted.

Thus you may log it this way:

**Accurate Logging**
See also

- mod_log_config
Apache Module mod_buffer

**Description:** Support for request buffering

**Status:** Extension

**Module Identifier:** buffer_module

**Source File:** mod_buffer.c

**Compatibility:** Available in Apache 2.3 and later

**Summary**

This module provides the ability to buffer the input and output filter stacks.

Under certain circumstances, content generators might create content in small chunks. In order to promote memory reuse, in memory chunks are always 8k in size, regardless of the size of the chunk itself. When many small chunks are generated by a request, this can create a large memory footprint while the request is being processed, and an unnecessarily large amount of data on the wire. The addition of a buffer collapses the response into the fewest chunks possible.

When httpd is used in front of an expensive content generator, buffering the response may allow the backend to complete processing and release resources sooner, depending on how the backend is designed.

The buffer filter may be added to either the input or the output filter stacks, as appropriate, using the `SetInputFilter`, `SetOutputFilter`, `AddOutputFilter` or `AddOutputFilterByType` directives.

**Using buffer with mod_include**

AddOutputFilterByType INCLUDES;BUFFER text/html
The buffer filters read the request/response into RAM and then repack the request/response into the fewest memory buckets possible, at the cost of CPU time. When the request/response is already efficiently packed, buffering the request/response could cause the request/response to be slower than not using a buffer at all. These filters should be used with care, and only where necessary.

Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

Filters
BufferSize Directive

Description: Maximum size in bytes to buffer by the buffer filter

Syntax: BufferSize integer

Default: BufferSize 131072

Context: server config, virtual host, directory, .htaccess

Status: Extension

Module: mod_buffer

The BufferSize directive specifies the amount of data in bytes that will be buffered before being read from or written to each request. The default is 128 kilobytes.
Apache mod_cache

URI

Extension

cache_module

mod_cache.c

Allow  Deny

mod_cache

mod_cache_disk

mod_mem_cache

mod_proxy  ProxyPass

Bugfix checklist
<table>
<thead>
<tr>
<th>mod cache disk</th>
<th>mod mem cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheRoot</td>
<td>CacheRoot</td>
</tr>
<tr>
<td>CacheDirLevels</td>
<td>CacheDirLevels</td>
</tr>
<tr>
<td>CacheDirLength</td>
<td>CacheDirLength</td>
</tr>
<tr>
<td>CacheMinFileSize</td>
<td>CacheMinFileSize</td>
</tr>
<tr>
<td>CacheMaxFileSize</td>
<td>CacheMaxFileSize</td>
</tr>
<tr>
<td>MCacheSize</td>
<td>MCacheSize</td>
</tr>
<tr>
<td>MCacheMaxObjectCount</td>
<td>MCacheMaxObjectCount</td>
</tr>
<tr>
<td>MCacheMinObjectSize</td>
<td>MCacheMinObjectSize</td>
</tr>
<tr>
<td>MCacheMaxObjectSize</td>
<td>MCacheMaxObjectSize</td>
</tr>
<tr>
<td>MCacheRemovalAlgorithm</td>
<td>MCacheRemovalAlgorithm</td>
</tr>
<tr>
<td>MCacheMaxStreamingBuffer</td>
<td>MCacheMaxStreamingBuffer</td>
</tr>
</tbody>
</table>
Sample httpd.conf

# Sample Cache Configuration

LoadModule cache_module modules/mod_cache.so

<IfModule mod_cache.c>
    # LoadModule cache_disk_module modules/mod_cache_disk.so
    # If you want to use mod_cache_disk instead of mod_mem_cache,
    # uncomment the line above and comment out the LoadModule line below.
    <IfModule mod_cache_disk.c>
        CacheRoot c:/cacheroot
        CacheEnable disk /
        CacheDirLevels 5
        CacheDirLength 3
    </IfModule>
</IfModule>

LoadModule mem_cache_module modules/mod_mem_cache.so

<IfModule mod_mem_cache.c>
    CacheEnable mem /
    MCacheSize 4096
    MCacheMaxObjectCount 100
    MCacheMinObjectSize 1
    MCacheMaxObjectSize 2048
</IfModule>

# When acting as a proxy, don't cache the list of security updates
CacheDisable http://security.update.server/update-list/
</IfModule>
<table>
<thead>
<tr>
<th>CacheDefaultExpire</th>
<th>seconds</th>
<th>3600 (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension</td>
<td></td>
<td></td>
</tr>
<tr>
<td>mod_cache</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CacheDefaultExpire: 86400

CacheMaxExpire: (expiry) (last-mod)
Add an X-Cache-Detail header to the response.

CacheDetailHeader on|off

CacheDetailHeader off

.,.,.htaccess

Extension

mod_cache

Available in Apache 2.3.9 and later
URL

CacheDisable *url-string*

, 

Extension

mod_cache

CacheDisable /local_files
CacheEnable mod_cache url-string URL

cache_type cache_type mem mod_mem_cache
mod_cache cache_type disk mod_cache_disk
mod_cache cache_type fd mod_cache
mod_mem_cache

() CacheEnable URL

CacheEnable mem /manual
CacheEnable fd /images
CacheEnable disk /

url-string

# Cache proxied url's
CacheEnable disk /

# Cache FTP-proxied url's
CacheEnable disk ftp://

# Cache content from www.apache.org
CacheEnable disk http://www.apache.org/
Add an X-Cache header to the response.

CacheHeader on|off
CacheHeader off

.htaccess
Extension
mod_cache

Available in Apache 2.3.9 and later
CacheIgnoreCacheControl

<table>
<thead>
<tr>
<th>CacheIgnoreCacheControl</th>
<th>On</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheIgnoreCacheControl</td>
<td>Off</td>
<td></td>
</tr>
</tbody>
</table>

Extension

mod_cache

Cache-Control: no-cache Pragma: no-store

CacheIgnoreCacheControl

CacheIgnoreCacheControl On

CacheIgnoreCacheControl On  no-cache

CacheIgnoreCacheControl On

- CacheStorePrivate
- CacheStoreNoStore
RFC 2616 hop-by-hop HTTP

**CacheIgnoreHeaders**

- Connection
- Keep-Alive
- Proxy-Authenticate
- Proxy-Authorization
- TE
- Trailers
- Transfer-Encoding
- Upgrade

1

```
CacheIgnoreHeaders Set-Cookie
```

2

```
CacheIgnoreHeaders None
```
<table>
<thead>
<tr>
<th>Expires</th>
<th>CacheIgnore</th>
</tr>
</thead>
</table>

To view the full text, please use the PDF viewer.
Last Modified
CacheIgnoreNoLastMod On|Off
CacheIgnoreNoLastMod Off
,
Extension
mod_cache

Last-Modified (Last-Modified) (expiry)
CacheIgnoreNoLastMod On
CacheIgnoreQueryString:

- CacheIgnoreQueryString: Off
- CacheIgnoreQueryString: On

Extension:

- mod_cache

CacheIgnoreQueryString: On
<table>
<thead>
<tr>
<th>CacheIgnoreURLSessionIdentifiers</th>
<th>Ignore defined session identifiers encoded in the URL when caching</th>
</tr>
</thead>
<tbody>
<tr>
<td>identifier</td>
<td>CacheIgnoreURLSessionIdentifiers identifier [identifier] ...</td>
</tr>
<tr>
<td>None</td>
<td>CacheIgnoreURLSessionIdentifiers None</td>
</tr>
<tr>
<td>mod_cache</td>
<td>Extension</td>
</tr>
</tbody>
</table>
Override the base URL of reverse proxied cache keys.

CacheKeyBaseURL  URL
CacheKeyBaseURL  http://example.com

Extension
mod_cache
Available in Apache 2.3.9 and later
<table>
<thead>
<tr>
<th>LastModified (expiry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheLastModifiedFactor float</td>
</tr>
<tr>
<td>CacheMaxExpire</td>
</tr>
</tbody>
</table>

```

CacheLastModifiedFactor 0.1

, ,

Extension

mod_cache

---

Last-Modified (expiry)

CacheLastModifiedFactor

expiry-period = time-since-last-modified-date * factor
expiry-date = current-date + expiry-period

10

factor 0.1 10*0.1 = 1

3:00pm + 1hour = 4:00pm

CacheMaxExpire

CacheLastModifiedFactor 0.5
```
<table>
<thead>
<tr>
<th>Enable the thundering herd lock.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheLock on</td>
</tr>
<tr>
<td>CacheLock off</td>
</tr>
<tr>
<td>Extension</td>
</tr>
<tr>
<td>mod_cache</td>
</tr>
<tr>
<td>Available in Apache 2.2.15 and later</td>
</tr>
</tbody>
</table>
Set the maximum possible age of a cache lock.

CacheLockMaxAge integer

CacheLockMaxAge 5

Extension

mod_cache
Set the lock path directory.

CacheLockPath directory

CacheLockPath /tmp/mod_cache-lock

Extension

mod_cache
<table>
<thead>
<tr>
<th>CacheMaxExpire</th>
<th>HTTP</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheMaxExpire 86400 ()</td>
<td></td>
</tr>
<tr>
<td>CacheMaxExpire 604800</td>
<td></td>
</tr>
</tbody>
</table>
CacheMinExpire seconds
CacheMinExpire 0

Extension
mod_cache

HTTP

CacheMinExpire 3600
<p>| <strong>Run the cache from the quick handler.</strong> |
| <strong>CacheQuickHandler on/off</strong> |
| <strong>CacheQuickHandler on</strong> |
| <strong>Extension</strong> |
| <strong>mod_cache</strong> |
| <strong>Apache HTTP Server 2.3.3 and later</strong> |</p>
<table>
<thead>
<tr>
<th>CacheStaleOnError</th>
<th>on/off</th>
</tr>
</thead>
</table>

Serve stale content in place of 5xx responses.

,. .,.htaccess

Extension

mod_cache

Available in Apache 2.3.9 and later
<table>
<thead>
<tr>
<th></th>
<th>Attempt to cache responses that the server reports as expired</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheStoreExpired On</td>
<td>On</td>
</tr>
<tr>
<td>CacheStoreExpired Off</td>
<td>Off</td>
</tr>
</tbody>
</table>

.,. .htaccess
Extension
mod_cache
Cache-Control: no-store

**CacheStoreNoCache** On  no-store

RFC 2616  no-store  " () "

- **CacheIgnoreCacheControl**
- **CacheStorePrivate**
| private                      |
| CacheStorePrivate On|Off |
| CacheStorePrivate Off       |

- **CacheIgnoreCacheControl**
- **CacheStoreNoStore**
Apache mod_cache_disk

<table>
<thead>
<tr>
<th>URI</th>
<th>Extension</th>
<th>cache_disk_module</th>
<th>mod_cache_disk.c</th>
</tr>
</thead>
</table>

mod_cache_disk

URI

htcacheclean

mod_cache_disk    mod_cache
<table>
<thead>
<tr>
<th>CacheDirLength</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheDirLength</td>
<td>2</td>
</tr>
</tbody>
</table>

Extension

| mod_cache_disk |

CacheDirLength

<table>
<thead>
<tr>
<th>CacheDirLevels</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheDirLength</td>
</tr>
<tr>
<td>20</td>
</tr>
</tbody>
</table>

```
CacheDirLevels

CacheDirLevels : 2

Extension

mod_cache_disk

CacheDirLevels : CacheDirLength

CacheDirLevels* CacheDirLength 20
CacheMinFileSize

<table>
<thead>
<tr>
<th>CacheMinFileSize bytes</th>
<th>CacheMinFileSize 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mod_cache_disk</td>
</tr>
</tbody>
</table>

CacheMinFileSize 64
The minimum size (in bytes) of the document to read and be cached before sending the data downstream.

, , .htaccess

Extension

mod_cache_disk

Documentation not yet translated. Please see English version of document.
CacheReadTime:
The minimum time (in milliseconds) that should elapse while reading before data is sent downstream.

, . . . .htaccess
Extension: mod_cache_disk

Documentation not yet translated. Please see English version of document.
CacheRoot directory
,
Extension
mod_cache_disk

CacheRoot
c:/cacheroot
Apache Module mod_cache_socache

**Description:** Shared object cache (socache) based storage module for the HTTP caching filter.

**Status:** Extension

**Module Identifier:** cache_socache_module

**Source File:** mod_cache_socache.c

**Summary**

`mod_cache_socache` implements a shared object cache (socache) based storage manager for `mod_cache`.

The headers and bodies of cached responses are combined, and stored underneath a single key in the shared object cache. A number of implementations of shared object caches are available to choose from.

Multiple content negotiated responses can be stored concurrently, however the caching of partial content is not yet supported by this module.

```
# Turn on caching
CacheSocache shmcb
CacheSocacheMaxSize 102400
<Location "/foo">
  CacheEnable socache
</Location>

# Fall back to the disk cache
CacheSocache shmcb
CacheSocacheMaxSize 102400
<Location "/foo">
  CacheEnable socache
  CacheEnable disk
```
Note:
mod_cache_socache requires the services of mod_cache, which must be loaded before mod_cache_socache.

Bugfix checklist
httpd changelog
Known issues
Report a bug

See also
mod_cache
mod_cache_disk
Caching Guide
**Description:** The shared object cache implementation to use

**Syntax:** CacheSocache  *type[:args]*

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_cache_socache

**Compatibility:** Available in Apache 2.4.5 and later

The CacheSocache directive defines the name of the shared object cache implementation to use, followed by optional arguments for that implementation. A number of implementations of shared object caches are available to choose from.

CacheSocache  *shmcb*
The `CacheSocacheMaxSize` directive sets the maximum size, in bytes, for the combined headers and body of a document to be considered for storage in the cache. The larger the headers that are stored alongside the body, the smaller the body may be.

The `mod_cache_socache` module will only attempt to cache responses that have an explicit content length, or that are small enough to be written in one pass. This is done to allow the `mod_cache_disk` module to have an opportunity to cache responses larger than those cacheable within `mod_cache_socache`.

```
CacheSocacheMaxSize 102400
```
**Description:** The maximum time (in seconds) for a document to be placed in the cache

**Syntax:** `CacheSocacheMaxTime seconds`

**Default:** `CacheSocacheMaxTime 86400`

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_cache_socache

**Compatibility:** Available in Apache 2.4.5 and later

The `CacheSocacheMaxTime` directive sets the maximum freshness lifetime, in seconds, for a document to be stored in the cache. This value overrides the freshness lifetime defined for the document by the HTTP protocol.

```
CacheSocacheMaxTime 86400
```
**Description:** The minimum time (in seconds) for a document to be placed in the cache

**Syntax:** CacheSocacheMinTime  seconds

**Default:** CacheSocacheMinTime  600

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_cache_socache

**Compatibility:** Available in Apache 2.4.5 and later

The **CacheSocacheMinTime** directive sets the amount of seconds beyond the freshness lifetime of the response that the response should be cached for in the shared object cache. If a response is only stored for its freshness lifetime, there will be no opportunity to revalidate the response to make it fresh again.

CacheSocacheMinTime  600
**CacheSocacheReadSize**

**Description:** The minimum size (in bytes) of the document to read and be cached before sending the data downstream.

**Syntax:** CacheSocacheReadSize  bytes

**Default:** CacheSocacheReadSize  0

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_cache_socache

**Compatibility:** Available in Apache 2.4.5 and later

The **CacheSocacheReadSize** directive sets the minimum amount of data, in bytes, to be read from the backend before the data is sent to the client. The default of zero causes all data read of any size to be passed downstream to the client immediately as it arrives. Setting this to a higher value causes the disk cache to buffer at least this amount before sending the result to the client. This can improve performance when caching content from a slow reverse proxy.

This directive only takes effect when the data is being saved to the cache, as opposed to data being served from the cache.

**CacheSocacheReadSize  102400**
### CacheSocacheReadTime

**Description:** The minimum time (in milliseconds) that should elapse while reading before data is sent downstream.

**Syntax:** CacheSocacheReadTime *milliseconds*

**Default:** CacheSocacheReadTime 0

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_cache_socache

**Compatibility:** Available in Apache 2.4.5 and later

---

The **CacheSocacheReadTime** directive sets the minimum amount of elapsed time that should pass before making an attempt to send data downstream to the client. During the time period, data will be buffered before sending the result to the client. This can improve performance when caching content from a reverse proxy.

The default of zero disables this option.

This directive only takes effect when the data is being saved to the cache, as opposed to data being served from the cache. It is recommended that this option be used alongside the **CacheSocacheReadSize** directive to ensure that the server does not buffer excessively should data arrive faster than expected.

```ini
CacheSocacheReadTime 1000
```
Apache Module mod_cern_meta

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>CERN httpd metafile semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>cern_meta_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_cern_meta.c</td>
</tr>
</tbody>
</table>

**Summary**

Emulate the CERN HTTPD Meta file semantics. Meta files are HTTP headers that can be output in addition to the normal range of headers for each file accessed. They appear rather like the Apache .asis files, and are able to provide a crude way of influencing the Expires: header, as well as providing other curiosities. There are many ways to manage meta information, this one was chosen because there is already a large number of CERN users who can exploit this module.

More information on the [CERN metafile semantics](https://cern.ch) is available.

**Bugfix checklist**

- [httpd changelog](https://httpd.apache.org/docs/2.4/changelog.html)
- [Known issues](https://httpd.apache.org/changes.html)
- [Report a bug](https://httpd.apache.org/report-a-bug.html)

**See also**

- [mod_headers](https://httpd.apache.org/docs/2.4/mod/mod_headers.html)
- [mod_asis](https://httpd.apache.org/docs/2.4/mod/mod_asis.html)
**Description:** Name of the directory to find CERN-style meta information files

**Syntax:** `MetaDir directory`

**Default:** `MetaDir .web`

**Context:** server config, virtual host, directory, .htaccess

**Override:** Indexes

**Status:** Extension

**Module:** `mod_cern_meta`

Specifies the name of the directory in which Apache can find meta information files. The directory is usually a 'hidden' subdirectory of the directory that contains the file being accessed. Set to "." to look in the same directory as the file:

```plaintext
MetaDir .
```

Or, to set it to a subdirectory of the directory containing the files:

```plaintext
MetaDir .meta
```
**Description:** Activates CERN meta-file processing

**Syntax:** `MetaFiles on|off`

**Default:** `MetaFiles off`

**Context:** server config, virtual host, directory, .htaccess

**Override:** Indexes

**Status:** Extension

**Module:** `mod_cern_meta`

Turns on/off Meta file processing on a per-directory basis.
### MetaSuffix

**Description:** File name suffix for the file containing CERN-style meta information.

**Syntax:**

```
MetaSuffix suffix
```

**Default:**

```
MetaSuffix .meta
```

**Context:**

server config, virtual host, directory, .htaccess

**Override:**

Indexes

**Status:**

Extension

**Module:**

mod_cern_meta

Specifies the file name suffix for the file containing the meta information. For example, the default values for the two directives will cause a request to DOCUMENT_ROOT/somedir/index.html to look in DOCUMENT_ROOT/somedir/.web/index.html.meta and will use its contents to generate additional MIME header information.

**Example:**

```
MetaSuffix .meta
```
Apache mod_cgi

:__ CGI
:__ Base
:__ cgi_module
:__ mod_cgi.c

cgi-script CGI
ScriptAlias

Apache CGI

Unix MPM

MIME application/x-httpd-cgi cgi-script
MIME

Bugfix checklist

httpd changelog
Known issues
Report a bug

AcceptPathInfo
Options  ExecCGI
ScriptAlias
AddHandler
CGI  ID
CGI
CGI

PATH_INFO
AcceptPathInfo off
mod_cgi (URI /more NOT FOUND AcceptPathInfo

REMOTE_HOST
HostnameLookups on ( off ) DNS

REMOTE_IDENT
IdentityCheck on ider

REMOTE_USER
CGI


%% [time] request-line
%% HTTP-status CGI-script-filename

2:

%%error
error-message

()
ScriptLog

<table>
<thead>
<tr>
<th>CGI</th>
<th>ScriptLog file-path</th>
<th>,</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base</td>
<td>mod_cgi, mod_cgid</td>
<td></td>
</tr>
</tbody>
</table>

ScriptLog CGI

ServerRoot

ScriptLog logs/cgi_log

User

CGI
<table>
<thead>
<tr>
<th>PUT</th>
<th>POST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ScriptLogBuffer</strong></td>
<td>bytes</td>
</tr>
<tr>
<td><strong>ScriptLogBuffer</strong></td>
<td>1024</td>
</tr>
<tr>
<td>,</td>
<td></td>
</tr>
<tr>
<td><strong>Base</strong></td>
<td></td>
</tr>
<tr>
<td><strong>mod_cgi, mod_cgid</strong></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CGI</td>
</tr>
<tr>
<td>-----</td>
<td>-----</td>
</tr>
<tr>
<td>ScriptLogLength</td>
<td>bytes</td>
</tr>
<tr>
<td>ScriptLogLength</td>
<td>10385760</td>
</tr>
<tr>
<td>,</td>
<td></td>
</tr>
<tr>
<td>Base</td>
<td></td>
</tr>
<tr>
<td>mod_cgi, mod_cgid</td>
<td></td>
</tr>
</tbody>
</table>
Apache mod_cgid

Bugfix checklist

httpd changelog
Known issues
Report a bug
mod_cgi

CGI_ID
The length of time to wait for more output from the CGI program

CGIScriptTimeout time[s|ms]

value of Timeout directive when unset

, , .htaccess

Base

mod_cgid

CGIScriptTimeout defaults to zero in releases 2.4 and earlier
ScriptSock `file-path`

ScriptSock `logs/cgisock`

Base

mod_cgid

ScriptSock `/var/run/cgid.sock`
Apache Module mod_charset_lite

**Description:** Specify character set translation or recoding

**Status:** Extension

**Module Identifier:** charset_lite_module

**Source File:** mod_charset_lite.c

**Summary**

`mod_charset_lite` allows the server to change the character set of responses before sending them to the client. In an EBCDIC environment, Apache always translates HTTP protocol content (e.g. response headers) from the code page of the Apache process locale to ISO-8859-1, but not the body of responses. In any environment, `mod_charset_lite` can be used to specify that response bodies should be translated. For example, if files are stored in EBCDIC, then `mod_charset_lite` can translate them to ISO-8859-1 before sending them to the client.

This module provides a small subset of configuration mechanisms implemented by Russian Apache and its associated mod_charset.
Invalid character set names

The character set name parameters of `CharsetSourceEnc` and `CharsetDefault` must be acceptable to the translation mechanism used by APR on the system where `mod_charset_lite` is deployed. These character set names are not standardized and are usually not the same as the corresponding values used in http headers. Currently, APR can only use `iconv(3)`, so you can easily test your character set names using the `iconv(1)` program, as follows:

```
iconv -f charsetsourceenc-value -t charsetdefault-value
```

Mismatch between character set of content and translation rules

If the translation rules don't make sense for the content, translation can fail in various ways, including:

- The translation mechanism may return a bad return code, and the connection will be aborted.
- The translation mechanism may silently place special characters (e.g., question marks) in the output buffer when it cannot translate the input buffer.
**CharsetDefault** directive specifies the charset that content in the associated container should be translated to.

The value of the `charset` argument must be accepted as a valid character set name by the character set support in **APR**. Generally, this means that it must be supported by `iconv`.

**Example**

```xml
<Directory "/export/home/trawick/apacheinst/htdocs/convert">
  CharsetSourceEnc UTF-16BE
  CharsetDefault ISO-8859-1
</Directory>
```

Specifying the same charset for both **CharsetSourceEnc** and **CharsetDefault** disables translation. The charset need not match the charset of the response, but it must be a valid charset on the system.
**CharsetOptions** directive configures certain behaviors of *mod_charset_lite*. *Option* can be one of

**ImplicitAdd | NoImplicitAdd**

The ImplicitAdd keyword specifies that *mod_charset_lite* should implicitly insert its filter when the configuration specifies that the character set of content should be translated. If the filter chain is explicitly configured using the *AddOutputFilter* directive, NoImplicitAdd should be specified so that *mod_charset_lite* doesn't add its filter.

**TranslateAllMimeTypes | NoTranslateAllMimeTypes**

Normally, *mod_charset_lite* will only perform translation on a small subset of possible mimetypes. When the TranslateAllMimeTypes keyword is specified for a given configuration section, translation is performed without regard for mimetype.
CharsetSourceEnc

Description: Source charset of files
Syntax: CharsetSourceEnc charset
Context: server config, virtual host, directory, .htaccess
Override: FileInfo
Status: Extension
Module: mod_charset_lite

The **CharsetSourceEnc** directive specifies the source charset of files in the associated container.

The value of the *charset* argument must be accepted as a valid character set name by the character set support in APR. Generally, this means that it must be supported by iconv.

**Example**

```apache
<Directory "/export/home/trawick/apacheinst/htdocs/convert"
  CharsetSourceEnc UTF-16BE
  CharsetDefault ISO-8859-1
</Directory>
```

The character set names in this example work with the iconv translation support in Solaris 8.

Specifying the same charset for both **CharsetSourceEnc** and **CharsetDefault** disables translation. The charset need not match the charset of the response, but it must be a valid charset on the system.
Apache Module mod_data

**Description:** Convert response body into an RFC2397 data URL

**Status:** Extension

**Module Identifier:** data_module

**Source File:** mod_data.c

**Compatibility:** Available in Apache 2.3 and later

### Summary

This module provides the ability to convert a response into an RFC2397 data URL.

Data URLs can be embedded inline within web pages using something like the `mod_include` module, to remove the need for clients to make separate connections to fetch what may potentially be many small images. Data URLs may also be included into pages generated by scripting languages such as PHP.

#### An example of a data URL

```
data:image/gif;base64,R0lGODdhMAAwAPAAAAAAAP///ywAAAAAMAAwAAAC8IyPqcvt3wCcDkiLc7C0qwyGHHsWpjQu5yqmcYsapyuvUULvONnOZtfzgFzByTB10Qgx0R0TqQQejhRnz0fkVJ+5YiUqrXF5Y5fKh/DeUnC5yLWGsEbtLiOspa/TPg7JpJHxyendzWTBfX0cxOnKPjgBzi4diinWdKF8kJdJfycQ7ZVeGejmJlZeGl9i2icVqaNvaiLT6F5iJ90m6mvuTS40K05M0vDk0Q4XUtxVkozrcd3i9uiesF81M10lIcR7lEewwcLp7tuNkM3uNna3F2JQFo97Vriy/XL4/f1cf5WzXyym7PHhhx4dbgYKAAA7
```

The filter takes no parameters, and can be added to the filter stack using the `SetOutputFilter` directive, or any of the directives supported by the `mod_filter` module.

### Configuring the filter

```
<Location "/data/images">
```

SetOutputFilter DATA
</Location>

Bugfix checklist

- httpd changelog
- Known issues
- Report a bug

See also

- Filters

Copyright 2017 The Apache Software Foundation.
Licensed under the Apache License, Version 2.0.
Apache mod_dav

: (WebDAV)
: Extension
dav_module
mod_dav.c

1 2

WebDAV ("")

Bugfix checklist

httpd changelog
Known issues
Report a bug

DavLockDB
LimitXMLRequestBody
WebDAV Resources
mod_dav  httpd.conf:

Dav On

DAV DAV

DAV DavLockDB  httpd.conf:

DavLockDB /usr/local/apache2/var/DavLock

Apache  User

<LIMIT>  <LOCATION> DAV
LimitXMLRequestBody

DavLockDB /usr/local/apache2/var/DavLock

<DIRECTORY /usr/local/apache2/htdocs/foo>
  Require all granted
  Dav On
  
  AuthType Basic
  AuthName DAV
  AuthUserFile user.passwd
  
  <LIMITEXCEPT GET POST OPTIONS>
    Require user admin
  </LIMITEXCEPT>
</DIRECTORY>
DAV

DAV

WebDAV

mod_dav

Group

)

mod_dav

DavDepthInfinity

PROPFIND

DAV
mod_dav (PHP CGI)

URL	DAV

Alias /phparea /home/gstein/php_files
Alias /php-source /home/gstein/php_files
<Location /php-source>
    Dav On
    ForceType text/plain
</Location>

http://example.com/phparea PHP
http://example.com/php-source DAV
Dav:

WebDAV HTTP

Dav On

Provider-name

Dav Off

Extension

mod_dav

WebDAV HTTP

<Location /foo>
  Dav On
</Location>

On mod_dav_fs filesystem

WebDAV
PROPFIND, Depth: Infinity
DavDepthInfinity on|off
DavDepthInfinity off
, ,
Extension
mod_dav

'Depth: Infinity' PROPFD

denial-of-service
DAV

DavMinTimeout  seconds
DavMinTimeout  0
Extension
mod_dav

DAV

(600)

<Location /MSWord>
    DavMinTimeout 600
</Location>
Apache mod_dav_fs

mod_dav

Extension
dav_fs_module
mod_dav_fs.c

mod_dav

Dav filesystem

filesystem mod_dav

Bugfix checklist
httpd changelog
Known issues
Report a bug

mod_dav
### DavLockDB

<table>
<thead>
<tr>
<th>DAV</th>
</tr>
</thead>
<tbody>
<tr>
<td>DavLockDB file-path</td>
</tr>
<tr>
<td>Extension</td>
</tr>
<tr>
<td>mod_dav_fs</td>
</tr>
</tbody>
</table>

SDBM

DavLockDB logs/DavLock

---

Copyright 2017 The Apache Software Foundation.  
[Apache License, Version 2.0](https://www.apache.org/licenses/LICENSE-2.0).
Apache mod_dav_lock

mod_dav

mod_dav Extension

dav_lock_module

mod_dav_lock.c

2.1

mod_dav API

subversion mod_dav_svn

mod_dav_fs

mod_dav_lock DavGenericLockDB

ap_lookup_provider

Bugfix checklist

httpd changelog
Known issues
Report a bug
mod_dav
DavGenericLockDB

mod_dav_lock

SDBM

DavGenericLockDB var/DavLock

Apache

ServerRoot var/

User

DavLock

Copyright 2017 The Apache Software Foundation.
Apache License, Version 2.0
Apache Module mod_dbd

**Description:** Manages SQL database connections

**Status:** Extension

**Module Identifier:** dbd_module

**Source File:** mod_dbd.c

**Compatibility:** Version 2.1 and later

**Summary**

mod_dbd manages SQL database connections using [APR](https://apr.apache.org). It provides database connections on request to modules requiring SQL database functions, and takes care of managing databases with optimal efficiency and scalability for both threaded and non-threaded MPMs. For details, see the [APR website](https://apr.apache.org) and this overview of the [Apache DBD Framework](https://httpd.apache.org/docs/current/mod/mod_dbd.html) by its original developer.

**Bugfix checklist**

- [httpd changelog](https://httpd.apache.org/docs/current/changelog.html)
- [Known issues](https://httpd.apache.org/docs/current/bug.html)
- [Report a bug](https://httpd.apache.org/lists.html?tech-bugs)

**See also**

- [Password Formats](https://httpd.apache.org/docs/current/security/password.html)
This module manages database connections, in a manner optimised for the platform. On non-threaded platforms, it provides a persistent connection in the manner of classic LAMP (Linux, Apache, Mysql, Perl/PHP/Python). On threaded platform, it provides an altogether more scalable and efficient connection pool, as described in this article at ApacheTutor. Note that mod_dbd supersedes the modules presented in that article.
To connect to your database, you'll need to specify a driver, and connection parameters. These vary from one database engine to another. For example, to connect to mysql, do the following:

```
DBDriver mysql
DBDParams host=localhost,dbname=pony,user=shetland,pass=appaloosa
```

You can then use this connection in a variety of other modules, including `mod rewrite`, `mod authn dbd`, and `mod lua`. Further usage examples appear in each of those modules' documentation.

See `DBDParams` for connection string information for each of the supported database drivers.
mod_dbd exports five functions for other modules to use. The API is as follows:

typedef struct {
    apr_dbd_t *handle;
    apr_dbd_driver_t *driver;
    apr_hash_t *prepared;
} ap_dbd_t;

/* Export functions to access the database */
/* acquire a connection that MUST be explicitly closed.
 * Returns NULL on error
 */
AP_DECLARE(ap_dbd_t*) ap_dbd_open(apr_pool_t*, server_rec*);

/* release a connection acquired with ap_dbd_open */
AP_DECLARE(void) ap_dbd_close(server_rec*, ap_dbd_t*);

/* acquire a connection that will have the lifetime of a request
 * and MUST NOT be explicitly closed. This is the preferred function for most apps.
 */
AP_DECLARE(ap_dbd_t*) ap_dbd_acquire(request_rec*);

/* acquire a connection that will have the lifetime of a connection
 * and MUST NOT be explicitly closed. Return NULL on error.
 */
AP_DECLARE(ap_dbd_t*) ap_dbd_cacquire(conn_rec*);

/* Prepare a statement for use by a client */
AP_DECLARE(void) ap_dbd_prepare(server_rec*, const char*, const char*);

/* Also export them as optional functions for
APR_DECLARE_OPTIONAL_FN(ap_dbd_t*, ap_dbd_open, (apr_pool_t*, server_rec*)); */
APR_DECLARE_OPTIONAL_FN(void, ap_dbd_close, server_rec*, ap_dbd_t*);
APR_DECLARE_OPTIONAL_FN(ap_dbd_t*, ap_dbd_acquire, request_rec*);
APR_DECLARE_OPTIONAL_FN(ap_dbd_t*, ap_dbd_cacquire, conn_rec*);
APR_DECLARE_OPTIONAL_FN(void, ap_dbd_prepare, server_rec*, const char*, const char*);
**mod_dbd** supports SQL prepared statements on behalf of modules that may wish to use them. Each prepared statement must be assigned a name (label), and they are stored in a hash: the prepared field of an ap_dbd_t. Hash entries are of type apr_dbd_prepared_t and can be used in any of the apr_dbd prepared statement SQL query or select commands.

It is up to dbd user modules to use the prepared statements and document what statements can be specified in httpd.conf, or to provide their own directives and use ap_dbd_prepare.

---

**Caveat**

When using prepared statements with a MySQL database, it is preferred to set reconnect to 0 in the connection string as to avoid errors that arise from the MySQL client reconnecting without properly resetting the prepared statements. If set to 1, any broken connections will be attempted fixed, but as mod_dbd is not informed, the prepared statements will be invalidated.
Any web/database application needs to secure itself against SQL injection attacks. In most cases, Apache DBD is safe, because applications use prepared statements, and untrusted inputs are only ever used as data. Of course, if you use it via third-party modules, you should ascertain what precautions they may require.

However, the *FreeTDS* driver is inherently *unsafe*. The underlying library doesn't support prepared statements, so the driver emulates them, and the untrusted input is merged into the SQL statement.

It can be made safe by *untainting* all inputs: a process inspired by Perl's taint checking. Each input is matched against a regexp, and only the match is used, according to the Perl idiom:

```perl
$untrusted =~ /([a-z]+)/;
$trusted = $1;
```

To use this, the untainting regexps must be included in the prepared statements configured. The regexp follows immediately after the % in the prepared statement, and is enclosed in curly brackets {}. For example, if your application expects alphanumeric input, you can use:

"SELECT foo FROM bar WHERE input = %s"

with other drivers, and suffer nothing worse than a failed query. But with FreeTDS you'd need:

"SELECT foo FROM bar WHERE input = %{{([A-Za-z0-9]+)}}s"

Now anything that doesn't match the regexp's $1 match is discarded, so the statement is safe.
An alternative to this may be the third-party ODBC driver, which offers the security of genuine prepared statements.
**Description:** Keepalive time for idle connections

**Syntax:** `DBDExptime time-in-seconds`

**Default:** `DBDExptime 300`

**Context:** server config, virtual host

**Status:** Extension

**Module:** `mod_dbd`

Set the time to keep idle connections alive when the number of connections specified in DBDKeep has been exceeded (threaded platforms only).
**Description:** Execute an SQL statement after connecting to a database

**Syntax:** `DBDInitSQL "SQL statement"

**Context:** server config, virtual host

**Status:** Extension

**Module:** `mod_dbd`

Modules, that wish it, can have one or more SQL statements executed when a connection to a database is created. Example usage could be initializing certain values or adding a log entry when a new connection is made to the database.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Maximum sustained number of connections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>DBDKeep  <em>number</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>DBDKeep  2</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_dbd</td>
</tr>
</tbody>
</table>

Set the maximum number of connections per process to be sustained, other than for handling peak demand (threaded platforms only).
**Description:** Maximum number of connections

**Syntax:** DBDMax *number*

**Default:** DBDMax 10

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_dbd

Set the hard maximum number of connections per process (threaded platforms only).
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Minimum number of connections</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>DBDMin <em>number</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>DBDMin 1</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_dbd</td>
</tr>
</tbody>
</table>

Set the minimum number of connections per process (threaded platforms only).
**Description:** Parameters for database connection

**Syntax:** DBDParams

\[param1=value1[,param2=value2]\]

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_dbd

As required by the underlying driver. Typically this will be used to pass whatever cannot be defaulted amongst username, password, database name,hostname and port number for connection.

Connection string parameters for current drivers include:

**FreeTDS (for MSSQL and SyBase)**

username, password, appname, dbname, host, charset, lang, server

**MySQL**

host, port, user, pass, dbname, sock, flags, fldsz, group, reconnect

**Oracle**

user, pass, dbname, server

**PostgreSQL**

The connection string is passed straight through to PQconnectdb

**SQLite2**

The connection string is split on a colon, and part1:part2 is used as sqlite_open(part1, atoi(part2), NULL)

**SQLite3**

The connection string is passed straight through to sqlite3_open

**ODBC**
datasource, user, password, connect, ctimeout, stimeout, access, txmode, bufsize
**Description:** Whether to use persistent connections

**Syntax:** `DBDPersist On|Off`

**Context:** server config, virtual host

**Status:** Extension

**Module:** `mod_dbd`

If set to Off, persistent and pooled connections are disabled. A new database connection is opened when requested by a client, and closed immediately on release. This option is for debugging and low-usage servers.

The default is to enable a pool of persistent connections (or a single LAMP-style persistent connection in the case of a non-threaded server), and should almost always be used in operation.

Prior to version 2.2.2, this directive accepted only the values 0 and 1 instead of Off and On, respectively.
**Description:** Define an SQL prepared statement

**Syntax:**
```
DBDPrepareSQL "SQL statement" label
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_dbd

For modules such as authentication that repeatedly use a single SQL statement, optimum performance is achieved by preparing the statement at startup rather than every time it is used. This directive prepares an SQL statement and assigns it a label.
**Description:** Specify an SQL driver

**Syntax:**

```
DBDriver name
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_dbd

Selects an apr_dbd driver by name. The driver must be installed on your system (on most systems, it will be a shared object or dll). For example, `DBDriver mysql` will select the MySQL driver in `apr_dbd_mysql.so`.
Apache mod_deflate

mod_deflate DEFLATE

Bugfix checklist

httpd changelog
Known issues
Report a bug

Filters
AddOutputFilterByType DEFLATE text/html text/plain text/xml

<Location />
  # Insert filter
  SetOutputFilter DEFLATE

  # Netscape 4.x has some problems...
  BrowserMatch ^Mozilla/4 gzip-only-text/html

  # Netscape 4.06-4.08 have some more problems
  BrowserMatch ^Mozilla/4\.[0][678] no-gzip

  # MSIE masquerades as Netscape, but it is fine
  # BrowserMatch \bMSIE !no-gzip !gzip-only-text/html

  # Don't compress images
  SetEnvIfNoCase Request_URI \(.?:gif|jpe?g|png)\$ no-gzip dont-vary

  # Make sure proxies don't deliver the wrong content
  Header append Vary User-Agent env=!dont-vary
</Location>
Output Compression

DEFLATE

SetOutputFilter DEFLATE

MIME

AddOutputFilterByType DEFLATE text/html

<Directory "/your-server-root/manual">
    AddOutputFilterByType DEFLATE text/html
</Directory>

BrowserMatch only-text/html

BrowserMatch ^Mozilla/4 gzip-only-text/html
BrowserMatch ^Mozilla/4\.[0678] no-gzip
BrowserMatch \bMSIE !no-gzip !gzip-only-text/html

User-Agent Netscape Navigator 4.x
    4.06, 4.07, 4.08 html

3 BrowserMatch "Mozilla/4"
SetEnv force-gzip accept-encoding

mod_deflate gzip inflate/uncompress:
  SetOutputFilter AddOutputFilter INFLATE

<Location /dav-area>
  ProxyPass http://example.com/
  SetOutputFilter INFLATE
</Location>

e.example.com gzip

mod_deflate gzip
  AddInputFilter DEFLATE

<Location /dav-area>
  SetInputFilter DEFLATE
</Location>

Content-Encoding: gzip

Content-Length
mod_deflate  Vary: Accept-Encoding HTTP
Accept-Encoding

User-Agent  Vary
DEFLATE

Header append Vary User-Agent

(HTTP)  Vary *

Header set Vary *
zlib
   DeflateBufferSize value
   DeflateBufferSize 8096
   ,
   Extension
   mod_deflate
DeflateCompressionLevel value

Zlib

Extension

mod_deflate

This directive is available since Apache 2.0.45

DeflateCompressionLevel

1() 9()
DeflateFilterNote

DeflateFilterNote ratio

LogFormat "%r %b (%{ratio}n) %{User-agent}i" deflate
CustomLog logs/deflate_log deflate

**Input**

**Output**

**Ratio**

\[
\frac{\text{Output}}{\text{Input}} \times 100
\]

DeflateFilterNote Input instream
DeflateFilterNote Output outstream
DeflateFilterNote Ratio ratio

LogFormat "%r %{outstream}n/%{instream}n (%{ratio}n%%)"
deflate
CustomLog logs/deflate_log deflate
- mod log config
Maximum size of inflated request bodies
DeflateInflateLimitRequestBody
None, but LimitRequestBody applies after deflation
,.,.htaccess
Extension
mod_deflate
2.4.10 and later
<table>
<thead>
<tr>
<th>Max number of times the inflation ratio for request bodies can be crossed</th>
<th>DeflateInflateRatioBurst value</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>.htaccess</td>
<td>Extension</td>
</tr>
<tr>
<td>mod_deflate</td>
<td>2.4.10 and later</td>
</tr>
</tbody>
</table>
Maximum inflation ratio for request bodies

DeflateInflateRatioLimit value

200

.htaccess

Extension

mod_deflate

2.4.10 and later
DeflateMemLevel

zlib

DeflateMemLevel value

DeflateMemLevel 9

,zlib

Extension

mod_deflate

DeflateMemLevel zlib (1 9)

)
DeflateWindowSize

__Zlib

DeflateWindowSize value

DeflateWindowSize 15

Extension

mod_deflate

```
DeflateWindowSize zlib (: zlib )

(: 2

```
Apache Module mod_dialup

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

**Status:** Experimental

**Module Identifier:** dialup_module

**Source File:** mod_dialup.c

**Summary**

It is a module that sends static content at a bandwidth rate limit, defined by the various old modem standards. So, you can browse your site with a 56k V.92 modem, by adding something like this:

```
<Location "/mysite">
    ModemStandard "V.92"
</Location>
```

Previously to do bandwidth rate limiting modules would have to block an entire thread, for each client, and insert sleeps to slow the bandwidth down. Using the new suspend feature, a handler can get callback N milliseconds in the future, and it will be invoked by the Event MPM on a different thread, once the timer hits. From there the handler can continue to send data to the client.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Modem standard to simulate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>ModemStandard</td>
</tr>
<tr>
<td></td>
<td>V.21</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>directory</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_dialup</td>
</tr>
</tbody>
</table>

Specify what modem standard you wish to simulate.

```html
<Location "/mysite">
    ModemStandard "V.26bis"
</Location>
```
**Apache mod_dir**

<table>
<thead>
<tr>
<th></th>
<th>Base</th>
<th>dir_module</th>
<th>mod_dir.c</th>
</tr>
</thead>
</table>

- `mod_dir` function

```c
index.html
```

```
http://servername/foo/dirname URL
```

```
http://servername/foo/dirname/
```
Toggle how this module responds when another handler is configured:
- DirectoryCheckHandler On|Off
- DirectoryCheckHandler Off
.
., .htaccess
.
Indexes
.
Base
.
mod_dir
.
Available in 2.4.8 and later. Releases prior to 2.4 implicitly act as if "DirectoryCheckHandler ON" was specified.
DirectoryIndex local-url [local-url] ...
DirectoryIndex index.html
..., .htaccess
Indexes
Base
mod_dir

URL

DirectoryIndex index.html

http://myserver/docs/
http://myserver/docs/index.html URL

DirectoryIndex index.html index.txt /cgi-bin/index.pl

index.html index.txt CGI
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DirectoryIndexRedirect</td>
<td>Configures an external redirect for directory indexes.</td>
</tr>
<tr>
<td></td>
<td><code>on</code></td>
</tr>
<tr>
<td></td>
<td><strong>DirectoryIndexRedirect off</strong></td>
</tr>
</tbody>
</table>

- `.htaccess`
- `Indexes`
- `Base`
- `mod_dir`
- Available in version 2.3.14 and later
DirectorySlash On|Off
DirectorySlash On
.,.htaccess
Indexes
Base
mod_dir
2.0.51

# see security warning below!
<Location /some/path>
  DirectorySlash Off
  SetHandler some-handler
</Location>

DirectoryIndex (index.html) URL
Define a default URL for requests that don't map to a file

, , .htaccess

Base

mod_dir

Documentation not yet translated. Please see English version of document.
Apache mod_dumpio

mod_dumpio Apache

SSL () SSL

Apache
Apache
<table>
<thead>
<tr>
<th>DumpIOInput</th>
<th>On</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>DumpIOInput</td>
<td>Off</td>
<td></td>
</tr>
</tbody>
</table>

**Extension**

*mod_dumpio*

**DumpIOInput** Apache 2.1.3

DumpIOInput On
<table>
<thead>
<tr>
<th>DumpIOOoutput</th>
<th>On</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>DumpIOOoutput</td>
<td>Off</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extension</th>
</tr>
</thead>
<tbody>
<tr>
<td>mod_dumpio</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DumpIOOoutput</th>
<th>Apache 2.1.3</th>
</tr>
</thead>
</table>

---

DumpIOOoutput On

---

Copyright 2017 The Apache Software Foundation. [Apache License, Version 2.0](https://www.apache.org/licenses/LICENSE-2.0)
Apache mod_echo

 Experimental
 echo_module
 mod_echo.c
ProtocolEcho

ProtocolEcho On

Experimental

mod_echo
Apache mod_env

:  CGI SSI
:  Base
:  env_module
:  mod_env.c

Bugfix checklist

httpd changelog
Known issues
Report a bug
PassEnv *env-variable* [env-variable] ...

, , , .htaccess

FileInfo

Base

mod_env

httpd  CGI  SSI

PassEnv LD_LIBRARY_PATH
SetEnv env-variable value

, , , .htaccess

FileInfo

Base

mod_env

CGI  SSI

SetEnv SPECIAL_PATH /foo/bin
UnsetEnv `env-variable` [`env-variable`] ...

, , .htaccess
FileInfo
Base
mod_env

CGI  SSI

UnsetEnv LD_LIBRARY_PATH
Apache Module mod_example_hooks

<table>
<thead>
<tr>
<th>Description:</th>
<th>Illustrates the Apache module API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Experimental</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>example_hooks_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_example_hooks.c</td>
</tr>
</tbody>
</table>

Summary

The files in the modules/examples directory under the Apache distribution directory tree are provided as an example to those that wish to write modules that use the Apache API.

The main file is mod_example_hooks.c, which illustrates all the different callback mechanisms and call syntaxes. By no means does an add-on module need to include routines for all of the callbacks - quite the contrary!

The example module is an actual working module. If you link it into your server, enable the "example-hooks-handler" handler for a location, and then browse to that location, you will see a display of some of the tracing the example module did as the various callbacks were made.
To include the example_hooks module in your server, follow the steps below:

1. Run `configure` with `--enable-example-hooks` option.
2. Make the server (run "make").

To add another module of your own:

A. `cp modules/examples/mod_example_hooks.c modules/new_module/mod_myexample.c`
B. Modify the file.
C. Create `modules/new_module/config.m4`.
   1. Add `APACHE_MODPATH_INIT(new_module)`.
   2. Copy `APACHE_MODULE` line with "example_hooks" from `modules/examples/config.m4`.
   3. Replace the first argument "example_hooks" with `myexample`.
   4. Replace the second argument with brief description of your module. It will be used in `configure --help`.
   5. If your module needs additional C compiler flags, linker flags or libraries, add them to `CFLAGS`, `LDFLAGS` and `LIBS` accordingly. See other `config.m4` files in modules directory for examples.
   6. Add `APACHE_MODPATH_FINISH`.
D. Create `module/new_module/Makefile.in`. If your module doesn't need special build instructions, all you need to have in that file is `include $(top_srcdir)/build/special.mk`.
E. Run `./buildconf` from the top-level directory.
F. Build the server with --enable-myexample
To activate the example_hooks module, include a block similar to the following in your `httpd.conf` file:

```
<Location "/example-hooks-info">
    SetHandler example-hooks-handler
</Location>
```

As an alternative, you can put the following into a `.htaccess` file and then request the file "test.example" from that location:

```
AddHandler example-hooks-handler ".example"
```

After reloading/restarting your server, you should be able to browse to this location and see the brief display mentioned earlier.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Demonstration directive to illustrate the Apache module API</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>Example</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_example_hooks</td>
</tr>
</tbody>
</table>

The **Example** directive just sets a demonstration flag which the example module's content handler displays. It takes no arguments. If you browse to an URL to which the example-hooks content-handler applies, you will get a display of the routines within the module and how and in what order they were called to service the document request. The effect of this directive one can observe under the point "Example directive declared here: YES/NO".
Apache mod_expires

Expires HTTP    Cache-Control    max-age

max-age (RFC 2616 section 14.9)    Cache-Control Header
ExpiresDefault  ExpiresByType

ExpiresDefault "<base> [plus] {<num> <type>}"  ExpiresByType type/encoding "<base> [plus] {<num> <type>}"

<base>:

- access
- now('access')
- modification

plus <num>  ( atoi())  <t

- years
- months
- weeks
- days
- hours
- minutes
- seconds

1  :

ExpiresDefault "access plus 1 month"
ExpiresDefault "access plus 4 weeks"
ExpiresDefault "access plus 30 days"

'<<num> <type>':

ExpiresByType text/html "access plus 1 month 15 days 2 hours"
ExpiresByType image/gif "modification plus 5 hours 3 minutes"

Expires
Expires
ExpiresActive On|Off
,,.htaccess
Indexes
Extension
mod_expires

Expires Cache-Control
ExpiresDefault ()
Expires Cache-Control (.htaccess)
ExpiresByType MIME-type <code>seconds</code>, .htaccess
Indexes
Extension
mod_expires

ExpiresActive On

# enable expirations
ExpiresActive On
# expire GIF images after a month in the client's cache
ExpiresByType image/gif A2592000
# HTML documents are good for a week from the
time they were changed
ExpiresByType text/html M604800

ExpiresActive On
ExpiresDefault <code>seconds</code>

Indexes

Extension

mod_expires
Apache mod_ext_filter

mod_ext_filter
Apache API  Apache

Bugfix checklist
httpd changelog
Known issues
Report a bug
```xml
<Directory "/export/home/trawick/apacheinst/htdocs/c">
  # core directive to cause the new filter to
  # be run on output
  SetOutputFilter c-to-html

  # mod_mime directive to set the type of .c
  # files to text/c
  AddType text/c .c

  # mod_ext_filter directive to set the debug
  # level just high enough to see a log message
  # per request showing the configuration in force
  ExtFilterOptions DebugLevel=1
</Directory>

<Location /gzipped>
  # core directive to cause the gzip filter to be
  # run on output
  SetOutputFilter gzip

  # mod_header directive to add
  # "Content-Encoding: gzip" header field
  Header set Content-Encoding gzip
</Location>
```
# mod_ext_filter directive to define a filter
# which runs everything through cat; cat doesn't
# modify anything; it just introduces extra pathlength
# and consumes more resources
ExtFilterDefine slowdown mode=output cmd=/bin/cat \  
preservescontentlength

<Location />
  # core directive to cause the slowdown filter to
  # be run several times on output
  #
  SetOutputFilter slowdown;slowdown;slowdown
</Location>

sed

# mod_ext_filter directive to define a filter which
# replaces text in the response
#
ExtFilterDefine fixtext mode=output intype=text/html \  
cmd="/bin/sed s/verdana/arial/g"

<Location />
  # core directive to cause the fixtext filter to
  # be run on output
  SetOutputFilter fixtext
</Location>

# Trace the data read and written by mod_deflate
# for a particular client (IP 192.168.1.31)
# experiencing compression problems.
# This filter will trace what goes into mod_deflate.
ExtFilterDefine tracebefore \  
cmd="/bin/tracefilter.pl /tmp/tracebefore" \  
EnableEnv=trace_this_client

# This filter will trace what goes after mod_deflate.
# Note that without the ftype parameter, the default
# filter type of AP_FTYPE_RESOURCE would cause the
# filter to be placed *before* mod_deflate in the filter
# chain. Giving it a numeric value slightly higher than
# AP_FTYPE_CONTENT_SET will ensure that it is placed
# after mod_deflate.
ExtFilterDefine traceafter \
   cmd="/bin/tracefilter.pl /tmp/traceafter" \
   EnableEnv=trace_this_client ftype=21

<Directory /usr/local/docs>
  SetEnvIf Remote_Addr 192.168.1.31 trace_this_client
  SetOutputFilter tracebefore;deflate;traceafter
</Directory>

#!/usr/local/bin/perl -w
use strict;

open(SAVE,">$ARGV[0]")
  or die "can't open $ARGV[0]: $?";

while (<STDIN>) {
  print SAVE $_;
  print $_; 
}

close(SAVE);
ExtFilterDefine *filtername* parameters

Extension
mod_ext_filter

ExtFilterDefine

*filtername*

SetOutput!

```
cmd=cmdline
    cmd=
        DOCUMENT_PATH_INFO, and
        QUERY_STRING_UNESCAPED
mode=mode
    mode=output ()
    Apache 2.1
intype=imt
    ( )
outtype=imt
    ( ) \n
PreservesContentLength
    PreservesContentLength content length
    content length
ftype=filtertype
    AP_FTYPE_RESOURCE
    AP_FTYPE_foo
```
disableenv=env

enableenv=env
ExtFilterOptions  mod_ext_filter

DebugLevel=n
  DebugLevel  mod_ext_filter
  DebugLevel=0
  DBGLVL_
    : Apache  core

LogStderr  |  NoLogStderr
  LogStderr
    (stdout)

ExtFilterOptions  LogStderr  DebugLevel=0

Apache
Apache Module mod_file_cache

**Description:** Caches a static list of files in memory
**Status:** Experimental
**Module Identifier:** file_cache_module
**Source File:** mod_file_cache.c

**Summary**

This module should be used with care. You can easily create a broken site using mod_file_cache, so read this document carefully.

*Caching* frequently requested files that change very infrequently is a technique for reducing server load. mod_file_cache provides two techniques for caching frequently requested static files. Through configuration directives, you can direct mod_file_cache to either open then mmap() a file, or to pre-open a file and save the file's open file handle. Both techniques reduce server load when processing requests for these files by doing part of the work (specifically, the file I/O) for serving the file when the server is started rather than during each request.

Notice: You cannot use this for speeding up CGI programs or other files which are served by special content handlers. It can only be used for regular files which are usually served by the Apache core content handler.

This module is an extension of and borrows heavily from the mod_mmap_static module in Apache 1.3.
mod_file_cache caches a list of statically configured files via MMapFile or CacheFile directives in the main server configuration.

Not all platforms support both directives. You will receive an error message in the server error log if you attempt to use an unsupported directive. If given an unsupported directive, the server will start but the file will not be cached. On platforms that support both directives, you should experiment with both to see which works best for you.

### MMapFile Directive

The MMapFile directive of mod_file_cache maps a list of statically configured files into memory through the system call mmap(). This system call is available on most modern Unix derivatives, but not on all. There are sometimes system-specific limits on the size and number of files that can be mmap()ed, experimentation is probably the easiest way to find out.

This mmap()ing is done once at server start or restart, only. So whenever one of the mapped files changes on the filesystem you have to restart the server (see the Stopping and Restarting documentation). To reiterate that point: if the files are modified in place without restarting the server you may end up serving requests that are completely bogus. You should update files by unlinking the old copy and putting a new copy in place. Most tools such as rdist and mv do this. The reason why this modules doesn't take care of changes to the files is that this check would need an extra stat() every time which is a waste and against the intent of I/O reduction.

### CacheFile Directive
The **CacheFile** directive of **mod_file_cache** opens an active handle or file descriptor to the file (or files) listed in the configuration directive and places these open file handles in the cache. When the file is requested, the server retrieves the handle from the cache and passes it to the sendfile() (or TransmitFile() on Windows), socket API.

This file handle caching is done once at server start or restart, only. So whenever one of the cached files changes on the filesystem you have to restart the server (see the **Stopping and Restarting** documentation). To reiterate that point: if the files are modified in place without restarting the server you may end up serving requests that are completely bogus. You should update files by unlinking the old copy and putting a new copy in place. Most tools such as rdist and mv do this.

**Note**

Don't bother asking for a directive which recursively caches all the files in a directory. Try this instead... See the **Include** directive, and consider this command:

```bash
find /www/htdocs -type f -print \
| sed -e 's/.*/mmapfile &/' > /www/conf/mmap.conf
```
**Description:** Cache a list of file handles at startup time

**Syntax:**

```
CacheFile file-path [file-path] ...
```

**Context:** server config

**Status:** Experimental

**Module:** mod_file_cache

The `CacheFile` directive opens handles to one or more files (given as whitespace separated arguments) and places these handles into the cache at server startup time. Handles to cached files are automatically closed on a server shutdown. When the files have changed on the filesystem, the server should be restarted to re-cache them.

Be careful with the `file-path` arguments: They have to literally match the filesystem path Apache's URL-to-filename translation handlers create. We cannot compare inodes or other stuff to match paths through symbolic links etc. because that again would cost extra `stat()` system calls which is not acceptable. This module may or may not work with filenames rewritten by `mod_alias` or `mod_rewrite`.

**Example**

```
CacheFile /usr/local/apache/htdocs/index.html
```
**Description:** Map a list of files into memory at startup time

**Syntax:**
```
MMapFile file-path [file-path] ...
```

**Context:** server config

**Status:** Experimental

**Module:** mod_file_cache

The **MMapFile** directive maps one or more files (given as whitespace separated arguments) into memory at server startup time. They are automatically unmapped on a server shutdown. When the files have changed on the filesystem at least a HUP or USR1 signal should be send to the server to re-mmap() them.

Be careful with the **file-path** arguments: They have to literally match the filesystem path Apache's URL-to-filename translation handlers create. We cannot compare inodes or other stuff to match paths through symbolic links etc. because that again would cost extra stat() system calls which is not acceptable. This module may or may not work with filenames rewritten by mod_alias or mod_rewrite.

**Example**
```
MMapFile /usr/local/apache/htdocs/index.html
```
Apache Module mod_filter

<table>
<thead>
<tr>
<th>Description:</th>
<th>Context-sensitive smart filter configuration module</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Base</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>filter_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_filter.c</td>
</tr>
<tr>
<td>Compatibility:</td>
<td>Version 2.1 and later</td>
</tr>
</tbody>
</table>

Summary

This module enables smart, context-sensitive configuration of output content filters. For example, apache can be configured to process different content-types through different filters, even when the content-type is not known in advance (e.g. in a proxy).

mod_filter works by introducing indirection into the filter chain. Instead of inserting filters in the chain, we insert a filter harness which in turn dispatches conditionally to a filter provider. Any content filter may be used as a provider to mod_filter; no change to existing filter modules is required (although it may be possible to simplify them).
In the traditional filtering model, filters are inserted unconditionally using AddOutputFilter and family. Each filter then needs to determine whether to run, and there is little flexibility available for server admins to allow the chain to be configured dynamically.

mod filter by contrast gives server administrators a great deal of flexibility in configuring the filter chain. In fact, filters can be inserted based on complex boolean expressions. This generalises the limited flexibility offered by AddOutputFilterByType.
In the traditional model, output filters are a simple chain from the content generator (handler) to the client. This works well provided the filter chain can be correctly configured, but presents problems when the filters need to be configured dynamically based on the outcome of the handler.

**Figure 1: The traditional filter model**

**Figure 2: The mod filter model**
*mod_filter* works by introducing indirection into the filter chain. Instead of inserting filters in the chain, we insert a filter harness which in turn dispatches conditionally to a filter provider. Any content filter may be used as a provider to *mod_filter*; no change to existing filter modules is required (although it may be possible to simplify them). There can be multiple providers for one filter, but no more than one provider will run for any single request.

A filter chain comprises any number of instances of the filter harness, each of which may have any number of providers. A special case is that of a single provider with unconditional dispatch: this is equivalent to inserting the provider filter directly into the chain.
There are three stages to configuring a filter chain with `mod_filter`. For details of the directives, see below.

**Declare Filters**

The `FilterDeclare` directive declares a filter, assigning it a name and filter type. Required only if the filter is not the default type `AP_FTYPE_RESOURCE`.

**Register Providers**

The `FilterProvider` directive registers a provider with a filter. The filter may have been declared with `FilterDeclare`; if not, `FilterProvider` will implicitly declare it with the default type `AP_FTYPE_RESOURCE`. The provider must have been registered with `ap_register_output_filter` by some module. The final argument to `FilterProvider` is an expression: the provider will be selected to run for a request if and only if the expression evaluates to true. The expression may evaluate HTTP request or response headers, environment variables, or the Handler used by this request. Unlike earlier versions, `mod_filter` now supports complex expressions involving multiple criteria with AND / OR logic (`&& / ||`) and brackets. The details of the expression syntax are described in the `ap_expr` documentation.

**Configure the Chain**

The above directives build components of a smart filter chain, but do not configure it to run. The `FilterChain` directive builds a filter chain from smart filters declared, offering the flexibility to insert filters at the beginning or end of the chain, remove a filter, or clear the chain.
mod_filter normally only runs filters on responses with HTTP status 200 (OK). If you want to filter documents with other response statuses, you can set the `filter-errordocs` environment variable, and it will work on all responses regardless of status. To refine this further, you can use expression conditions with `FilterProvider`. 
Upgrading from Apache HTTP Server 2.2

Configuration

The FilterProvider directive has changed from httpd 2.2: the match and dispatch arguments are replaced with a single but more versatile expression. In general, you can convert a match/dispatch pair to the two sides of an expression, using something like:

"dispatch = 'match'"

The Request headers, Response headers and Environment variables are now interpreted from syntax %{req:foo}, %{resp:foo} and %{env:foo} respectively. The variables %{HANDLER} and %{CONTENT_TYPE} are also supported.

Note that the match no longer support substring matches. They can be replaced by regular expression matches.
Server side Includes (SSI)
A simple case of replacing AddOutputFilterByType

FilterDeclare SSI
FilterProvider SSI INCLUDES "%(CONTENT_TYPE)~|^text/html|"
FilterChain SSI

Server side Includes (SSI)
The same as the above but dispatching on handler (classic SSI behaviour; .shtml files get processed).

FilterProvider SSI INCLUDES "%(HANDLER)"
FilterChain SSI

Emulating mod_gzip with mod_deflate
Insert INFLATE filter only if "gzip" is NOT in the Accept-Encoding header. This filter runs with ftype CONTENT_SET.

FilterDeclare gzip CONTENT_SET
FilterProvider gzip inflate "%(req:Accept-Encoding)~!/gzip/"
FilterChain gzip

Image Downsampling
Suppose we want to downsample all web images, and have filters for GIF, JPEG and PNG.

FilterProvider unpack jpeg_unpack "%(CONTENT_TYPE)~=image/jpeg'"
FilterProvider unpack gif_unpack "%(CONTENT_TYPE)~=image/gif'"
FilterProvider unpack png_unpack "%(CONTENT_TYPE)~=image/png'"
FilterProvider downsample downsample_file
FilterProtocol downsample "change=yes"

FilterProvider repack jpeg_pack "%{CONTENT_TYPE}
FilterProvider repack gif_pack "%{CONTENT_TYPE}
FilterProvider repack png_pack "%{CONTENT_TYPE}

<Location "/image-filter">
    FilterChain unpack downsample repack
</Location>
Historically, each filter is responsible for ensuring that whatever changes it makes are correctly represented in the HTTP response headers, and that it does not run when it would make an illegal change. This imposes a burden on filter authors to re-implement some common functionality in every filter:

- Many filters will change the content, invalidating existing content tags, checksums, hashes, and lengths.
- Filters that require an entire, unbroken response in input need to ensure they don't getbyteranges from a backend.
- Filters that transform output in a filter need to ensure they don't violate a Cache-Control: no-transform header from the backend.
- Filters may make responses uncachable.

`mod_filter` aims to offer generic handling of these details of filter implementation, reducing the complexity required of content filter modules. This is work-in-progress; the `FilterProtocol` implements some of this functionality for back-compatibility with Apache 2.0 modules. For httpd 2.1 and later, the `ap_register_output_filter_protocol` and `ap_filter_protocol` API enables filter modules to declare their own behaviour.

At the same time, `mod_filter` should not interfere with a filter that wants to handle all aspects of the protocol. By default (i.e. in the absence of any `FilterProtocol` directives), `mod_filter` will leave the headers untouched.

At the time of writing, this feature is largely untested, as modules in common use are designed to work with 2.0. Modules using it should test it carefully.
**AddOutputFilterByType**

**Description:**
assigns an output filter to a particular media-type

**Syntax:**
```
AddOutputFilterByType
filter[;filter...] media-type
[media-type] ...
```

**Context:**
server config, virtual host, directory, .htaccess

**Override:**
FileInfo

**Status:**
Base

**Module:**
mod_filter

**Compatibility:**
Had severe limitations before being moved to mod_filter in version 2.3.7

This directive activates a particular output filter for a request depending on the response media-type.

The following example uses the DEFLATE filter, which is provided by mod_deflate. It will compress all output (either static or dynamic) which is labeled as text/html or text/plain before it is sent to the client.

```
AddOutputFilterByType DEFLATE text/html text/plain
```

If you want the content to be processed by more than one filter, their names have to be separated by semicolons. It's also possible to use one AddOutputFilterByType directive for each of these filters.

The configuration below causes all script output labeled as text/html to be processed at first by the INCLUDES filter and then by the DEFLATE filter.

```
<Location "/cgi-bin/">
```

---

---
Options Includes
AddOutputFilterByType INCLUDES;DEFLATE
</Location>

See also
- AddOutputFilter
- SetOutputFilter
- filters
This configures an actual filter chain, from declared filters. 
FilterChain takes any number of arguments, each optionally preceded with a single-character control that determines what to do:

+filter-name
   Add filter-name to the end of the filter chain

@filter-name
   Insert filter-name at the start of the filter chain

-filter-name
   Remove filter-name from the filter chain

=filter-name
   Empty the filter chain and insert filter-name

!
   Empty the filter chain

filter-name
   Equivalent to +filter-name
### FilterDeclare

| **Description:** | Declare a smart filter |
| **Syntax:**     | FilterDeclare filter-name [type] |
| **Context:**    | server config, virtual host, directory, .htaccess |
| **Override:**   | Options |
| **Status:**     | Base |
| **Module:**     | mod_filter |

This directive declares an output filter together with a header or environment variable that will determine runtime configuration. The first argument is a `filter-name` for use in `FilterProvider`, `FilterChain` and `FilterProtocol` directives.

The final (optional) argument is the type of filter, and takes values of `ap_filter_type` - namely RESOURCE (the default), CONTENT_SET, PROTOCOL, TRANSCODE, CONNECTION or NETWORK.
**Description:** Deal with correct HTTP protocol handling

**Syntax:** FilterProtocol filter-name [provider-name] proto-flags

**Context:** server config, virtual host, directory, .htaccess

**Override:** Options

**Status:** Base

**Module:** mod_filter

This directs mod filter to deal with ensuring the filter doesn't run when it shouldn't, and that the HTTP response headers are correctly set taking into account the effects of the filter.

There are two forms of this directive. With three arguments, it applies specifically to a filter-name and a provider-name for that filter. With two arguments it applies to a filter-name whenever the filter runs any provider.

Flags specified with this directive are merged with the flags that underlying providers may have registerd with mod filter. For example, a filter may internally specify the equivalent of change=yes, but a particular configuration of the module can override with change=no.

proto-flags is one or more of

**change=yes | no**

  Specifies whether the filter changes the content, including possibly the content length. The "no" argument is supported in 2.4.7 and later.

**change=1:1**

  The filter changes the content, but will not change the content length
**byteranges=no**
The filter cannot work on byteranges and requires complete input

**proxy=no**
The filter should not run in a proxy context

**proxy=transform**
The filter transforms the response in a manner incompatible with the HTTP Cache-Control: no-transform header.

**cache=no**
The filter renders the output uncachable (eg by introducing randomised content changes)
**Description:** Register a content filter

**Syntax:**

```
FilterProvider filter-name provider-name expression
```

**Context:** server config, virtual host, directory, .htaccess

**Override:** Options

**Status:** Base

**Module:** mod_filter

This directive registers a *provider* for the smart filter. The provider will be called if and only if the *expression* declared evaluates to true when the harness is first called.

*provider-name* must have been registered by loading a module that registers the name with `ap_register_output_filter`.

*expression* is an `ap_expr`.

### See also

- [Expressions in Apache HTTP Server](#), for a complete reference and examples.
- [mod_include](#)
**Description:** Get debug/diagnostic information from `mod_filter`  

**Syntax:** `FilterTrace filter-name level`  

**Context:** server config, virtual host, directory  

**Status:** Base  

**Module:** `mod_filter`  

This directive generates debug information from `mod_filter`. It is designed to help test and debug providers (filter modules), although it may also help with `mod_filter` itself.

The debug output depends on the `level` set:

- **0 (default)**
  - No debug information is generated.

- **1**
  - `mod_filter` will record buckets and brigades passing through the filter to the error log, before the provider has processed them. This is similar to the information generated by `mod_diagnostics`.

- **2 (not yet implemented)**
  - Will dump the full data passing through to a tempfile before the provider. **For single-user debug only**; this will not support concurrent hits.
Apache mod_headers

HTTP

Extension

headers_module

mod_headers.c

RequestHeader  Apache 2.0
**mod_headers**

RequestHeader append MirrorID "mirror 12"
RequestHeader unset MirrorID

<table>
<thead>
<tr>
<th>MirrorID</th>
<th>MirrorID &quot;mirror 12&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
mod_headers
/
early
early
1. "TS" :

   Header echo ^TS

2. MyHeader :

   Header add MyHeader "%D %t"

   MyHeader: D=3775428 t=991424704447256

3. Joe :

   Header add MyHeader "Hello Joe. It took %D microseconds for Apache to serve this request."

   MyHeader: Hello Joe. It took D=3775428 microseconds for Apache to serve this request.

4. "MyRequestHeader"  MyHeader

   mod_setenvif

   SetEnvIf MyRequestHeader value HAVE_MyRequestHeader
   Header add MyHeader "%D %t mytext" env=HAVE_MyRequestHeader

   HTTP MyRequestHeader: value

   MyHeader: D=3775428 t=991424704447256 mytext
HTTP

Header [condition] set|append|add|unset|echo
  header [value] [early|env=[!]variable]

,,.htaccess
FileInfo
Extension
mod_headers

HTTP

  condition  onsuccess  always
  always      (2xx)

  set

append

add

()  

unset

echo

( header)
header

add, append, set  value  
value  
value :
<table>
<thead>
<tr>
<th>%</th>
<th>Universal Coordinated Time t=</th>
</tr>
</thead>
<tbody>
<tr>
<td>%t</td>
<td>FOOBAR</td>
</tr>
<tr>
<td>%D</td>
<td>SSL FOOBAR</td>
</tr>
<tr>
<td>% {FOOBAR}e</td>
<td>mod_ssl</td>
</tr>
<tr>
<td>% {FOOBAR}s</td>
<td>SSL FOOBAR</td>
</tr>
</tbody>
</table>

```
Header
env=!...)

Header

Header
```
HTTP

set

append

add

unset

( header)

value unset vi

Header

RequestHeader env=!

RequestHeader fixup
Apache Module mod_heartbeat

<table>
<thead>
<tr>
<th>Description:</th>
<th>Sends messages with server status to frontend proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Experimental</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>heartbeat_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_heartbeat</td>
</tr>
<tr>
<td>Compatibility:</td>
<td>Available in Apache 2.3 and later</td>
</tr>
</tbody>
</table>

**Summary**

mod_heartbeat sends multicast messages to a mod_heartmonitor listener that advertises the server's current connection count. Usually, mod_heartmonitor will be running on a proxy server with mod_lbmethod_heartbeat loaded, which allows ProxyPass to use the "heartbeat" lbmethod inside of ProxyPass.

mod_heartbeat itself is loaded on the origin server(s) that serve requests through the proxy server(s).

To use mod_heartbeat, mod_status and mod_watchdog must be either a static module or, if a dynamic module, must be loaded before mod_heartbeat.
Every 1 second, this module generates a single multicast UDP packet, containing the number of busy and idle workers. The packet is a simple ASCII format, similar to GET query parameters in HTTP.

An Example Packet

v=1&ready=75&busy=0

Consumers should handle new variables besides busy and ready, separated by '&', being added in the future.
HeartbeatAddress Directive

**Description:** Multicast address for heartbeat packets

**Syntax:** HeartbeatAddress *addr:port*

**Default:** disabled

**Context:** server config

**Status:** Experimental

**Module:** mod_heartbeat

The HeartbeatAddress directive specifies the multicast address to which mod_heartbeat will send status information. This address will usually correspond to a configured HeartbeatListen on a frontend proxy system.

HeartbeatAddress 239.0.0.1:27999
Apache Module mod_heartmonitor

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Centralized monitor for mod_heartbeat origin servers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>heartmonitor_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_heartmonitor.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.3 and later</td>
</tr>
</tbody>
</table>

**Summary**

mod_heartmonitor listens for server status messages generated by mod_heartbeat enabled origin servers and makes their status available to mod_lbmethod_heartbeat. This allows ProxyPass to use the "heartbeat" lbmethod inside of ProxyPass.

This module uses the services of mod_slotmem_shm when available instead of flat-file storage. No configuration is required to use mod_slotmem_shm.

To use mod_heartmonitor, mod_status and mod_watchdog must be either a static modules or, if a dynamic module, it must be loaded before mod_heartmonitor.
**Description:** multicast address to listen for incoming heartbeat requests

**Syntax:** `HeartbeatListen addr:port`

**Default:** disabled

**Context:** server config

**Status:** Experimental

**Module:** `mod_heartmonitor`

The `HeartbeatListen` directive specifies the multicast address on which the server will listen for status information from `mod_heartbeat`-enabled servers. This address will usually correspond to a configured `HeartbeatAddress` on an origin server.

```
HeartbeatListen 239.0.0.1:27999
```

This module is inactive until this directive is used.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Specifies the maximum number of servers that will be sending heartbeat requests to this server</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td><code>HeartbeatMaxServers number-of-servers</code></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td><code>HeartbeatMaxServers 10</code></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td><code>mod_heartmonitor</code></td>
</tr>
</tbody>
</table>

The **HeartbeatMaxServers** directive specifies the maximum number of servers that will be sending requests to this monitor server. It is used to control the size of the shared memory allocated to store the heartbeat info when `mod_slotmem_shm` is in use.
HeartbeatStorage Directive

**Description:** Path to store heartbeat data

**Syntax:**

```
HeartbeatStorage file-path
```

**Default:**

```
HeartbeatStorage logs/hb.dat
```

**Context:** server config

**Status:** Experimental

**Module:** mod_heartmonitor

The **HeartbeatStorage** directive specifies the path to store heartbeat data. This flat-file is used only when **mod_slotmem_shm** is not loaded.
Apache Module mod_http2

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Support for the HTTP/2 transport layer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>http2_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_http2.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in version 2.4.17 and later</td>
</tr>
</tbody>
</table>

**Summary**

This module provides HTTP/2 (RFC 7540) support for the Apache HTTP Server.

This module relies on [libnghttp2](http://libnghttp2.org) to provide the core http/2 engine.

You must enable HTTP/2 via `Protocols` in order to use the functionality described in this document. The HTTP/2 protocol does not require the use of encryption so two schemes are available: h2 (HTTP/2 over TLS) and h2c (HTTP/2 over TCP).

Two useful configuration schemes are:

**HTTP/2 in a VirtualHost context (TLS only)**

```
Protocols h2 http/1.1
```

Allows HTTP/2 negotiation (h2) via TLS ALPN in a secure `<VirtualHost>`. HTTP/2 preamble checking (Direct mode, see [H2Direct](http://httpd.apache.org/docs/2.4/mod/mod_http2.html)) is disabled by default for h2.

**HTTP/2 in a Server context (TLS and cleartext)**

```
Protocols h2 h2c http/1.1
```
Allows HTTP/2 negotiation (h2) via TLS ALPN for secure <VirtualHost>. Allows HTTP/2 cleartext negotiation (h2c) upgrading from an initial HTTP/1.1 connection or via HTTP/2 preamble checking (Direct mode, see H2Direct).

Refer to the official HTTP/2 FAQ for any doubt about the protocol.
HTTP/2 Dimensioning

Enabling HTTP/2 on your Apache Server has impact on the resource consumption and if you have a busy site, you may need to consider carefully the implications.

The first noticeable thing after enabling HTTP/2 is that your server processes will start additional threads. The reason for this is that HTTP/2 gives all requests that it receives to its own Worker threads for processing, collects the results and streams them out to the client.

In the current implementation, these workers use a separate thread pool from the MPM workers that you might be familiar with. This is just how things are right now and not intended to be like this forever. (It might be forever for the 2.4.x release line, though.) So, HTTP/2 workers, or shorter H2Workers, will not show up in mod_status. They are also not counted against directives such as ThreadsPerChild. However they take ThreadsPerChild as default if you have not configured something else via H2MinWorkers and H2MaxWorkers.

Another thing to watch out for is is memory consumption. Since HTTP/2 keeps more state on the server to manage all the open request, priorities for and dependencies between them, it will always need more memory than HTTP/1.1 processing. There are three directives which steer the memory footprint of a HTTP/2 connection: H2MaxSessionStreams, H2WindowSize and H2StreamMaxMemSize.

H2MaxSessionStreams limits the number of parallel requests that a client can make on a HTTP/2 connection. It depends on your site how many you should allow. The default is 100 which is plenty and unless you run into memory problems, I would keep it
this way. Most requests that browsers send are GETs without a body, so they use up only a little bit of memory until the actual processing starts.

**H2WindowSize** controls how much the client is allowed to send as body of a request, before it waits for the server to encourage more. Or, the other way around, it is the amount of request body data the server needs to be able to buffer. This is per request.

And last, but not least, **H2StreamMaxMemSize** controls how much response data shall be buffered. The request sits in a H2Worker thread and is producing data, the HTTP/2 connection tries to send this to the client. If the client does not read fast enough, the connection will buffer this amount of data and then suspend the H2Worker.

**Multiple Hosts and Misdirected Requests**

Many sites use the same TLS certificate for multiple virtual hosts. The certificate either has a wildcard name, such as '*.example.org' or carries several alternate names. Browsers using HTTP/2 will recognize that and reuse an already opened connection for such hosts.

While this is great for performance, it comes at a price: such vhosts need more care in their configuration. The problem is that you will have multiple requests for multiple hosts on the same TLS connection. And that makes renegotiation impossible, in face the HTTP/2 standard forbids it.

So, if you have several virtual hosts using the same certificate and want to use HTTP/2 for them, you need to make sure that all vhosts have exactly the same SSL configuration. You need the same protocol, ciphers and settings for client verification.

If you mix things, Apache httpd will detect it and return a special
response code, 421 Misdirected Request, to the client.

**Environment Variables**

This module can be configured to provide HTTP/2 related information as additional environment variables to the SSI and CGI namespace, as well as in custom log configurations (see `% {VAR_NAME}e`).

<table>
<thead>
<tr>
<th>Variable Name</th>
<th>Value Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP2</td>
<td>flag</td>
<td>HTTP/2 is being used.</td>
</tr>
<tr>
<td>H2PUSH</td>
<td>flag</td>
<td>HTTP/2 Server Push is enabled for this connection and also supported by the client.</td>
</tr>
<tr>
<td>H2_PUSH</td>
<td>flag</td>
<td>alternate name for H2PUSH</td>
</tr>
<tr>
<td>H2_PUSHED</td>
<td>string</td>
<td>empty or PUSHED for a request being pushed by the server.</td>
</tr>
<tr>
<td>H2_PUSHED_ON</td>
<td>number</td>
<td>HTTP/2 stream number that triggered the push of this request.</td>
</tr>
<tr>
<td>H2_STREAM_ID</td>
<td>number</td>
<td>HTTP/2 stream number of this request.</td>
</tr>
<tr>
<td>H2_STREAM_TAG</td>
<td>string</td>
<td>HTTP/2 process unique stream identifier, consisting of connection id and stream id separated by - .</td>
</tr>
</tbody>
</table>
**Description:** Determine file handling in responses

**Syntax:**

H2CopyFiles on|off

**Default:**

H2CopyFiles off

**Context:**

server config, virtual host, directory, .htaccess

**Status:**

Extension

**Module:**

mod_http2

**Compatibility:** Available in version 2.4.24 and later.

This directive influences how file content is handled in responses. When off, which is the default, file handles are passed from the request processing down to the main connection, using the usual Apache setaside handling for managing the lifetime of the file.

When set to on, file content is copied while the request is still being processed and the buffered data is passed on to the main connection. This is better if a third party module is injecting files with different lifetimes into the response.

An example for such a module is mod_wsgi that may place Python file handles into the response. Those files get close down when Python thinks processing has finished. That may be well before mod_http2 is done with them.
**Description:** H2 Direct Protocol Switch

**Syntax:** H2Direct on|off

**Default:** H2Direct on for h2c, off for h2 protocol

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_http2

This directive toggles the usage of the HTTP/2 Direct Mode. This should be used inside a `<VirtualHost>` section to enable direct HTTP/2 communication for that virtual host.

Direct communication means that if the first bytes received by the server on a connection match the HTTP/2 preamble, the HTTP/2 protocol is switched to immediately without further negotiation. This mode is defined in RFC 7540 for the cleartext (h2c) case. Its use on TLS connections not mandated by the standard.

When a server/vhost does not have h2 or h2c enabled via `Protocols`, the connection is never inspected for a HTTP/2 preamble. H2Direct does not matter then. This is important for connections that use protocols where an initial read might hang indefinitely, such as NNTP.

For clients that have out-of-band knowledge about a server supporting h2c, direct HTTP/2 saves the client from having to perform an HTTP/1.1 upgrade, resulting in better performance and avoiding the Upgrade restrictions on request bodies.

This makes direct h2c attractive for server to server communication as well, when the connection can be trusted or is secured by other means.
Example
H2Direct on
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Determine sending of 103 status codes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>H2EarlyHints on</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>H2EarlyHints off</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_http2</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in version 2.4.24 and later.</td>
</tr>
</tbody>
</table>

This setting controls if HTTP status 103 interim responses are forwarded to the client or not. By default, this is currently not the case since a range of clients still have trouble with unexpected interim responses.

When set to on, PUSH resources announced with H2PushResource will trigger an interim 103 response before the final response. The 103 response will carry Link headers that advise the preload of such resources.
Description: Maximum number of active streams per HTTP/2 session.

Syntax: H2MaxSessionStreams $n$

Default: H2MaxSessionStreams 100

Context: server config, virtual host

Status: Extension

Module: mod_http2

This directive sets the maximum number of active streams per HTTP/2 session (e.g. connection) that the server allows. A stream is active if it is not idle or closed according to RFC 7540.

Example

H2MaxSessionStreams 20
**Description:** Maximum number of seconds h2 workers remain idle until shut down.

**Syntax:**  
H2MaxWorkerIdleSeconds  n

**Default:**  
H2MaxWorkerIdleSeconds  600

**Context:**  
server config

**Status:**  
Extension

**Module:**  
mod_http2

This directive sets the maximum number of seconds a h2 worker may idle until it shuts itself down. This only happens while the number of h2 workers exceeds H2MinWorkers.

**Example**

H2MaxWorkerIdleSeconds  20
**Description:** Maximum number of worker threads to use per child process.

**Syntax:**

```
H2MaxWorkers n
```

**Context:** server config

**Status:** Extension

**Module:** mod_http2

This directive sets the maximum number of worker threads to spawn per child process for HTTP/2 processing. If this directive is not used, `mod_http2` will chose a value suitable for the mpm module loaded.

**Example**

```
H2MaxWorkers 20
```
**Description:** Minimal number of worker threads to use per child process.

**Syntax:** H2MinWorkers \( n \)

**Context:** server config

**Status:** Extension

**Module:** mod_http2

This directive sets the minimum number of worker threads to spawn per child process for HTTP/2 processing. If this directive is not used, `mod_http2` will choose a value suitable for the mpm module loaded.

**Example**

H2MinWorkers 10
H2ModernTLSOnly

**Description:** Require HTTP/2 connections to be "modern TLS" only

**Syntax:** H2ModernTLSOnly on|off

**Default:** H2ModernTLSOnly on

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_http2

**Compatibility:** Available in version 2.4.18 and later.

This directive toggles the security checks on HTTP/2 connections in TLS mode (https:). This can be used server wide or for specific `<VirtualHost>`s.

The security checks require that the TSL protocol is at least TLSv1.2 and that none of the ciphers listed in RFC 7540, Appendix A is used. These checks will be extended once new security requirements come into place.

The name stems from the [Security/Server Side TLS](https://www.mozilla.org) definitions at mozilla where "modern compatibility" is defined. Mozilla Firefox and other browsers require modern compatibility for HTTP/2 connections. As everything in OpSec, this is a moving target and can be expected to evolve in the future.

One purpose of having these checks in mod_http2 is to enforce this security level for all connections, not only those from browsers. The other purpose is to prevent the negotiation of HTTP/2 as a protocol should the requirements not be met.

Ultimately, the security of the TLS connection is determined by the server configuration directives for mod_ssl.

**Example**
**Description:** H2 Server Push Switch

**Syntax:**

```
H2Push on|off
```

**Default:**

```
H2Push on
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_http2

**Compatibility:** Available in version 2.4.18 and later.

This directive toggles the usage of the HTTP/2 server push protocol feature.

The HTTP/2 protocol allows the server to push other resources to a client when it asked for a particular one. This is helpful if those resources are connected in some way and the client can be expected to ask for it anyway. The pushing then saves the time it takes the client to ask for the resources itself. On the other hand, pushing resources the client never needs or already has is a waste of bandwidth.

Server pushes are detected by inspecting the Link headers of responses (see https://tools.ietf.org/html/rfc5988 for the specification). When a link thus specified has the `rel=preload` attribute, it is treated as a resource to be pushed.

Link headers in responses are either set by the application or can be configured via `mod_headers` as:

```plaintext
mod_headers example

<Location /index.html>
  Header add Link "</css/site.css>;rel=preload"
  Header add Link "</images/logo.jpg>;rel=preload"
</Location>
```
As the example shows, there can be several link headers added to a response, resulting in several pushes being triggered. There are no checks in the module to avoid pushing the same resource twice or more to one client. Use with care.

HTTP/2 server pushes are enabled by default. This directive allows it to be switch off on all resources of this server/virtual host.

Example
H2Push off

Last but not least, pushes happen only when the client signals its willingness to accept those. Most browsers do, some, like Safari 9, do not. Also, pushes also only happen for resources from the same authority as the original response is for.
This directive toggles the maximum number of HTTP/2 server pushes that are remembered per HTTP/2 connection. This can be used inside the `<VirtualHost>` section to influence the number for all connections to that virtual host.

The push diary records a digest (currently using a 64 bit number) of pushed resources (their URL) to avoid duplicate pushes on the same connection. These value are not persisted, so clients opening a new connection will experience known pushes again. There is ongoing work to enable a client to disclose a digest of the resources it already has, so the diary maybe initialized by the client on each connection setup.

If the maximum size is reached, newer entries replace the oldest ones. A diary entry uses 8 bytes, letting a default diary with 256 entries consume around 2 KB of memory.

A size of 0 will effectively disable the push diary.
This directive defines the priority handling of pushed responses based on the content-type of the response. This is usually defined per server config, but may also appear in a virtual host.

HTTP/2 server pushes are always related to a client request. Each such request/response pairs, or *streams* have a dependency and a weight, together defining the *priority* of a stream.

When a stream *depends* on another, say X depends on Y, then Y gets all bandwidth before X gets any. Note that this does not mean that Y will block X. If Y has no data to send, all bandwidth allocated to Y can be used by X.

When a stream has more than one dependant, say X1 and X2 both depend on Y, the *weight* determines the bandwidth allocation. If X1 and X2 have the same weight, they both get half of the available bandwidth. If the weight of X1 is twice as large as that for X2, X1 gets twice the bandwidth of X2.

Ultimately, every stream depends on the *root* stream which gets all the bandwidth available, but never sends anything. So all its bandwidth is distributed by weight among its children. Which either
have data to send or distribute the bandwidth to their own children. And so on. If none of the children have data to send, that bandwidth get distributed somewhere else according to the same rules.

The purpose of this priority system is to always make use of available bandwidth while allowing precedence and weight to be given to specific streams. Since, normally, all streams are initiated by the client, it is also the one that sets these priorities.

Only when such a stream results in a PUSH, gets the server to decide what the initial priority of such a pushed stream is. In the examples below, X is the client stream. It depends on Y and the server decides to PUSH streams P1 and P2 onto X.

The default priority rule is:

**Default Priority Rule**

H2PushPriority * After 16

which reads as 'Send a pushed stream of any content-type depending on the client stream with weight 16'. And so P1 and P2 will be send after X and, as they have equal weight, share bandwidth equally among themselves.

**Interleaved Priority Rule**

H2PushPriority text/css Interleaved 256

which reads as 'Send any CSS resource on the same dependency and weight as the client stream'. If P1 has content-type 'text/css', it will depend on Y (as does X) and its effective weight will be calculated as P1ew = Xw * (P1w / 256). With P1w being 256, this will make the effective weight the same as the weight of X. If both X and P1 have data to send, bandwidth will be allocated
to both equally.

With Pw specified as 512, a pushed, interleaved stream would get double the weight of X. With 128 only half as much. Note that effective weights are always capped at 256.

**Before Priority Rule**

H2PushPriority application/json Before

This says that any pushed stream of content type 'application/json' should be send out before X. This makes P1 dependent on Y and X dependent on P1. So, X will be stalled as long as P1 has data to send. The effective weight is inherited from the client stream. Specifying a weight is not allowed.

Be aware that the effect of priority specifications is limited by the available server resources. If a server does not have workers available for pushed streams, the data for the stream may only ever arrive when other streams have been finished.

Last, but not least, there are some specifics of the syntax to be used in this directive:

1. '* is the only special content-type that matches all others. 'image/*' will not work.

2. The default dependency is 'After'.

3. There are also default weights: for 'After' it is 16, 'interleaved' is 256.

**Shorter Priority Rules**

H2PushPriority application/json 32 # an After rule
H2PushPriority image/jpeg before # weight inherited
H2PushPriority text/css interleaved # weight 256 default
**Description:** Declares resources for early pushing to the client

**Syntax:**
H2PushResource [add] path [critical]

**Context:**
server config, virtual host, directory, .htaccess

**Status:**
Extension

**Module:**
mod_http2

**Compatibility:**
Available in version 2.4.24 and later.

When added to a directory/location HTTP/2 PUSHes will be attempted for all paths added via this directive. This directive can be used several times for the same location.

This directive pushes resources much earlier than adding Link headers via `mod_headers`. `mod_http2` announces these resources in a 103 Early Hints interim response to the client. That means that clients not supporting PUSH will still get early preload hints.

In contrast to setting Link response headers via `mod_headers`, this directive will only take effect on HTTP/2 connections.

By adding critical to such a resource, the server will give processing it more preference and send its data, once available, before the data from the main request.
**Description:** Serialize Request/Response Processing Switch

**Syntax:** H2SerializeHeaders on|off

**Default:** H2SerializeHeaders off

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_http2

This directive toggles if HTTP/2 requests shall be serialized in HTTP/1.1 format for processing by httpd core or if received binary data shall be passed into the request_recs directly.

Serialization will lower performance, but gives more backward compatibility in case custom filters/hooks need it.

**Example**

H2SerializeHeaders on
**Description:** Maximum amount of output data buffered per stream.

**Syntax:** H2StreamMaxMemSize bytes

**Default:** H2StreamMaxMemSize 65536

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_http2

This directive sets the maximum number of outgoing data bytes buffered in memory for an active streams. This memory is not allocated per stream as such. Allocations are counted against this limit when they are about to be done. Stream processing freezes when the limit has been reached and will only continue when buffered data has been sent out to the client.

**Example**

H2StreamMaxMemSize 128000
**Description:**
H2TLSCoolDownSecs *seconds*

**Syntax:**
H2TLSCoolDownSecs *seconds*

**Default:**
H2TLSCoolDownSecs 1

**Context:**
server config, virtual host

**Status:**
Extension

**Module:**
mod_http2

**Compatibility:**
Available in version 2.4.18 and later.

This directive sets the number of seconds of idle time on a TLS connection before the TLS write size falls back to small (~1300 bytes) length. This can be used server wide or for specific `<VirtualHost>`s.

See [H2TLSWarmUpSize](#) for a description of TLS warmup. H2TLSCoolDownSecs reflects the fact that connections may deteriorate over time (and TCP flow adjusts) for idle connections as well. It is beneficial to overall performance to fall back to the pre-warmup phase after a number of seconds that no data has been sent.

In deployments where connections can be considered reliable, this timer can be disabled by setting it to 0.

The following example sets the seconds to zero, effectively disabling any cool down. Warmed up TLS connections stay on maximum record size.

**Example**

```
H2TLSCoolDownSecs 0
```
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>This directive sets the number of bytes to be sent in small TLS records (~1300 bytes) until doing maximum sized writes (16k) on https: HTTP/2 connections. This can be used server wide or for specific <code>&lt;VirtualHost&gt;</code>s.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>H2TLSWarmUpSize <em>amount</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>H2TLSWarmUpSize 1048576</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_http2</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in version 2.4.18 and later.</td>
</tr>
</tbody>
</table>

Measurements by [google performance labs](https://www.google.com) show that best performance on TLS connections is reached, if initial record sizes stay below the MTU level, to allow a complete record to fit into an IP packet.

While TCP adjust its flow-control and window sizes, longer TLS records can get stuck in queues or get lost and need retransmission. This is of course true for all packets. TLS however needs the whole record in order to decrypt it. Any missing bytes at the end will stall usage of the received ones.

After a sufficient number of bytes have been send successfully, the TCP state of the connection is stable and maximum TLS record sizes (16 KB) can be used for optimal performance.

In deployments where servers are reached locally or over reliable connections only, the value might be decreased with 0 disabling any warmup phase altogether.

The following example sets the size to zero, effectively disabling...
any warmup phase.

Example

H2TLSWarmUpSize 0
**Description:** H2 Upgrade Protocol Switch

**Syntax:**

H2Upgrade on|off

**Default:**

H2Upgrade on for h2c, off for h2 protocol

**Context:**

server config, virtual host

**Status:**

Extension

**Module:**

mod_http2

This directive toggles the usage of the HTTP/1.1 Upgrade method for switching to HTTP/2. This should be used inside a `<VirtualHost>` section to enable Upgrades to HTTP/2 for that virtual host.

This method of switching protocols is defined in HTTP/1.1 and uses the "Upgrade" header (thus the name) to announce willingness to use another protocol. This may happen on any request of a HTTP/1.1 connection.

This method of protocol switching is enabled by default on cleartext (potential h2c) connections and disabled on TLS (potential h2), as mandated by RFC 7540.

Please be aware that Upgrades are only accepted for requests that carry no body. POSTs and PUTs with content will never trigger an upgrade to HTTP/2. See [H2Direct](#) for an alternative to Upgrade.

This mode only has an effect when h2 or h2c is enabled via the [Protocols](#).

**Example**

H2Upgrade on
Description: Size of Stream Window for upstream data.

Syntax: H2WindowSize bytes

Default: H2WindowSize 65535

Context: server config, virtual host

Status: Extension

Module: mod_http2

This directive sets the size of the window that is used for flow control from client to server and limits the amount of data the server has to buffer. The client will stop sending on a stream once the limit has been reached until the server announces more available space (as it has processed some of the data).

This limit affects only request bodies, not its meta data such as headers. Also, it has no effect on response bodies as the window size for those are managed by the clients.

Example

H2WindowSize 128000
Apache mod_ident

- RFC 1413 ident lookups
- Extension
- ident_module
- mod_ident.c
- Apache 2.1

RFC 1413

Bugfix checklist
- httpd changelog
- Known issues
- Report a bug

mod_log_config
| RFC 1413 | IdentityCheck On|Off |
| --- | --- |
| IdentityCheck Off | , |
| Extension | mod_ident |
| Apache 2.1 core | identd |
```
: Ident
: IdentityCheckTimeout seconds
: IdentityCheckTimeout 30
: ,
: Extension
: mod_ident

ident                      30
```
Apache Module mod_imagemap

**Description:** Server-side imagemap processing

**Status:** Base

**Module Identifier:** imagemap_module

**Source File:** mod_imagemap.c

**Summary**

This module processes .map files, thereby replacing the functionality of the imagemap CGI program. Any directory or document type configured to use the handler imap-file (using either **AddHandler** or **SetHandler**) will be processed by this module.

The following directive will activate files ending with .map as imagemap files:

```
AddHandler imap-file map
```

Note that the following is still supported:

```
AddType application/x-httpd-imap map
```

However, we are trying to phase out "magic MIME types" so we are deprecating this method.
The imagemap module adds some new features that were not possible with previously distributed imagemap programs.

- URL references relative to the Referer: information.
- Default `<base>` assignment through a new map directive `base`.
- No need for `imagemap.conf` file.
- Point references.
- Configurable generation of imagemap menus.
The lines in the imagemap files can have one of several formats:

```
directive value [x,y ...]
directive value "Menu text" [x,y ...]
directive value x,y ... "Menu text"
```

The directive is one of base, default, poly, circle, rect, or point. The value is an absolute or relative URL, or one of the special values listed below. The coordinates are $x, y$ pairs separated by whitespace. The quoted text is used as the text of the link if an imagemap menu is generated. Lines beginning with '#' are comments.

**Imagemap File Directives**

There are six directives allowed in the imagemap file. The directives can come in any order, but are processed in the order they are found in the imagemap file.

**base Directive**

Has the effect of `<base href="value">`. The non-absolute URLs of the map-file are taken relative to this value. The base directive overrides `ImapBase` as set in a `.htaccess` file or in the server configuration files. In the absence of an `ImapBase` configuration directive, base defaults to `http://server_name/`.

`base_uri` is synonymous with `base`. Note that a trailing slash on the URL is significant.

**default Directive**

The action taken if the coordinates given do not fit any of the poly, circle or rect directives, and there are no point directives. Defaults to `nocontent` in the absence of an
**.Imaging**

ImapDefault configuration setting, causing a status code of 204 No Content to be returned. The client should keep the same page displayed.

**poly Directive**

Takes three to one-hundred points, and is obeyed if the user selected coordinates fall within the polygon defined by these points.

**circle**

Takes the center coordinates of a circle and a point on the circle. Is obeyed if the user selected point is with the circle.

**rect Directive**

Takes the coordinates of two opposing corners of a rectangle. Obeyed if the point selected is within this rectangle.

**point Directive**

Takes a single point. The point directive closest to the user selected point is obeyed if no other directives are satisfied. Note that default will not be followed if a point directive is present and valid coordinates are given.

**Values**

The values for each of the directives can be any of the following:

**a URL**

The URL can be relative or absolute URL. Relative URLs can contain '..' syntax and will be resolved relative to the base value.

base itself will not be resolved according to the current value. A statement base mailto: will work properly, though.

**map**

Equivalent to the URL of the imagemap file itself. No
coordinates are sent with this, so a menu will be generated unless `ImapMenu` is set to none.

**menu**

Synonymous with map.

**referer**

Equivalent to the URL of the referring document. Defaults to `http://servername/` if no `Referer:` header was present.

**nocontent**

Sends a status code of 204 No Content, telling the client to keep the same page displayed. Valid for all but base.

**error**

Fails with a 500 Server Error. Valid for all but base, but sort of silly for anything but default.

### Coordinates

0,0 200,200

A coordinate consists of an `x` and a `y` value separated by a comma. The coordinates are separated from each other by whitespace. To accommodate the way Lynx handles imagemaps, should a user select the coordinate 0, 0, it is as if no coordinate had been selected.

### Quoted Text

"Menu Text"

After the value or after the coordinates, the line optionally may contain text within double quotes. This string is used as the text for the link if a menu is generated:

```html
<a href="http://example.com/">Menu text</a>
```

If no quoted text is present, the name of the link will be used
as the text:

```
<a href="http://example.com/">http://example.com</a>
```

If you want to use double quotes within this text, you have to write them as &quot;.
Comments are printed in a 'formatted' or 'semiformatted' menu. And can contain html tags. <hr>
base referer
poly map "Could I have a menu, please?" 0,0 0,10 10,10 10,0
rect .. 0,0 77,27 "the directory of the referer"
circle http://www.inetnebr.example.com/lincoln/feedback/ 195,0 305,27
rect another_file "in same directory as referer" 306,0 419,27
point http://www.zyzzyva.example.com/ 100,100
point http://www.tripod.example.com/ 200,200
rect mailto:nate@tripod.example.com 100,150 200,0 "Bugs?"
Referencing your mapfile

**HTML example**

```html
<a href="/maps/imagemap1.map">
    <img ismap src="/images/imagemap1.gif">
</a>
```

**XHTML example**

```html
<a href="/maps/imagemap1.map">
    <img ismap="ismap" src="/images/imagemap1.gif" />
</a>
```
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Default base for imagemap files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>`ImapBase map</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td><code>ImapBase http://servername/</code></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td><strong>Override:</strong></td>
<td>Indexes</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td><code>mod_imagemap</code></td>
</tr>
</tbody>
</table>

The **ImapBase** directive sets the default base used in the imagemap files. Its value is overridden by a base directive within the imagemap file. If not present, the base defaults to `http://servername/`.

**See also**

- [UseCanonicalName](#)
The **ImapDefault** directive sets the default default used in the imagemap files. Its value is overridden by a default directive within the imagemap file. If not present, the default action is `nocontent`, which means that a 204 No Content is sent to the client. In this case, the client should continue to display the original page.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Action if no coordinates are given when calling an imagemap</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>ImapMenu</td>
</tr>
<tr>
<td></td>
<td>none</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>ImapMenu formatted</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td><strong>Override:</strong></td>
<td>Indexes</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_imagemap</td>
</tr>
</tbody>
</table>

The **ImapMenu** directive determines the action taken if an imagemap file is called without valid coordinates.

**none**
If ImapMenu is none, no menu is generated, and the default action is performed.

**formatted**
A formatted menu is the simplest menu. Comments in the imagemap file are ignored. A level one header is printed, then an hrule, then the links each on a separate line. The menu has a consistent, plain look close to that of a directory listing.

**semiformatted**
In the semiformatted menu, comments are printed where they occur in the imagemap file. Blank lines are turned into HTML breaks. No header or hrule is printed, but otherwise the menu is the same as a formatted menu.

**unformatted**
Comments are printed, blank lines are ignored. Nothing is printed that does not appear in the imagemap file. All breaks and headers must be included as comments in the imagemap file. This gives you the most flexibility over the appearance of
your menus, but requires you to treat your map files as HTML instead of plaintext.
Apache mod_include

- html (Server Side Includes)
- Base
- include_module
- mod_include.c
- Apache 2.0

Bugfix checklist

httpd changelog
Known issues
Report a bug

Options
AcceptPathInfo

SSI
Server-Side Includes

Server-side includes .shtml

Apache

AddType text/html .shtml
AddOutputFilter INCLUDES .shtml

.shtml

Options .htaccess):

Options +Includes

server-parsed INCLUDES MIME
server-parsed-html text/x-server-parsed-html

Apache INCLUDES ( MIME

Tutorial on Server Side Includes.
SGML

<!-#element attribute=value attribute=value ... -->

(: value) (') ()

( -->) SSI

(: element)

<table>
<thead>
<tr>
<th>config</th>
<th>configure output formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>echo</td>
<td>print variables</td>
</tr>
<tr>
<td>exec</td>
<td>execute external programs</td>
</tr>
<tr>
<td>fsize</td>
<td>print size of a file</td>
</tr>
<tr>
<td>flastmod</td>
<td>print last modification time of a file</td>
</tr>
<tr>
<td>include</td>
<td>include a file</td>
</tr>
<tr>
<td>printenv</td>
<td>print all available variables</td>
</tr>
<tr>
<td>set</td>
<td>set a value of a variable</td>
</tr>
</tbody>
</table>

SSI  mod_include

config

echomsg *(Apache 2.1)*

echo

errmsg

sizefmt

)
timefmt
strftime(3)

echo

include SSIUndefinedEcho

var

encoding

echo encoding entity encoding
encoding var ISO-8859-1

exec
exec CGI

mod_

cgi

(%-) URL (/)

(ScriptAlias Option ExecCGI ) CGI

CGI PATH_INFO ( CGI include

<!--#exec cgi="/cgi-bin/example.cgi" -->
exec cgi include_virtual CGI

cgi include virtual

<--#include virtual="/cgi-bin/example.cgi?argument=value" -->

#include virtual #exec cgi #exec cmd

( #include virtual) Apache

Win32 suexec unix exec
unix suexec Win32 suexec unix:

<--#exec cmd="perl /path/to/perlscript arg1 arg2" -->

FSIZE

SIZEFORMAT

FILE

VIRTUAL

(\ ) URL-path (/)

FILEMOD

SIZEFORM

INCLUDE
(text/plain, text/html)

include

generic

file

virtual

<!--#include virtual="/cgi-bin/example.cgi?argument=value" -->

HTML CGI

KeptBodySize

GET

printenv

Apache 1.3.12

--->

set

var
value

<!--#set var="category" value="help" -->

```html

```
CGI echo if elif,
SSI
\texttt{echo, set}:
\[
\text{<!--#if expr="$a = \$test" -->}
\]
\[
\text{<!--set var="Zed" value="${REMOTE\_HOST}_${REQUEST\_METHOD}" -->}
\]
\[
\text{REMOTE\_HOST "X" REQUEST\_METHOD "Y" Zed "X\_Y"}
\]
\[
\text{DOCUMENT\_URI /foo/file.html "in foo"}
\]
\[
\text{/bar/file.html "in bar" "in neither"}
\]
\[
\text{<!--#if expr="\$DOCUMENT\_URI = "/foo/file.html" -->}
\]
\[
\text{in foo}
\]
\[
\text{<!--#elif expr="\$DOCUMENT\_URI = "/bar/file.html" -->}
\]
\[
\text{in bar}
\]
\[
\text{<!--#else -->}
\]
\[
\text{in neither}
\]
\[
\text{<!--#endif -->}
\]
if
    if

elif
else test_condition
endif

if test_condition:
    string
    string

-A string
    (: httpd) URL true
    SSIAccessEnable URL
    URL URL

Example
<!-#if expr="-A /private" -->
    Click <a href="/private">here</a> to access private information.
<!-#endif -->

string1 = string2
string1 == string2
string1 != string2

perl 5 == =
( == ==)
string1 < string2
string1 <= string2
string1 > string2
string1 >= string2
    string1 string2 ( strcmp(3) ) "100"
    "20"
    ( test_condition )
    test_condition
! test_condition
    test_condition
    test_condition1 && test_condition2
    test_condition1 test_condition2
    test_condition1 || test_condition2
    test_condition1 test_condition2
"
" !=" " !=" " &&" " !" 
    :

&& ||
    :

string1 string2 string1 string2
'string1  string2' string1  string2

•

• ( && || )
•

-DDEBUG_INCLUDE

/
/

«
mod\_include

include

mod\_include

include

SSIEndTag "%>"

• SSIStartTag
SSIErrorMsg

SSIErrorMsg "[an error occurred while processing this directive]"

SSIErrorMsg mod_include

<!--#config errmsg=message -->

SSIErrorMsg "<!-- Error -->"
Controls whether ETags are generated by the server.

SSIETag on|off
SSIETag off
.htaccess
Base
mod_include
Available in version 2.2.15 and later.
Controls whether Last-Modified headers are generated by the server.

SSLLastModified on|off
SSLLastModified off
.htaccess
Base
mod_include
Available in version 2.2.15 and later.
Enable compatibility mode for conditional expressions.

SSILegacyExprParser on|off

SSILegacyExprParser off

.htaccess

Base

mod_include

Available in version 2.3.13 and later.
mod_include include

()
SSITimeFormat formatstring
SSITimeFormat "%A, %d-%b-%Y %H:%M:%S %Z"

,..., .htaccess
All
Base
mod_include
2.0.30

DATE echo

<!--#config timefmt=formatstring -->

SSITimeFormat "%R, %B %d, %Y"

"22:26, June 14, 2002"
echo
SSIUndefinedEcho "string"
SSIUndefinedEcho "(none)"
.,.htaccess
All
Base
mod_include
2.0.34

"echo"  mod

SSIUndefinedEcho "<!-- undef -->"
XBitHack HTML

off

on  
  text/html html

full  
on

CGI  #include
Apache mod_info

mod_info

httpd.conf

<Location /server-info>
    SetHandler server-info
</Location>

<Location mod_access>
    <Location /server-info>
        SetHandler server-info
        Order deny,allow
        Deny from all
        Allow from yourcompany.com
    </Location>
</Location>

http://your.host.example.com/server-info
Security

mod_info

Apache

mod_authz_host

```
<Location /server-info>
    SetHandler server-info
    Order allow,deny
    # Allow access from server itself
    Allow from 127.0.0.1
    # Additionally, allow access from local workstation
    Allow from 192.168.1.17
</Location>
```
server-info
http://your.host.example.com/server-info?config

<?module-name>

?config

?hooks

?list

?server
mod_info

- Include, <IfModule>, <IfDefine>,
- ()
- .htaccess
- <Directory>
- mod_perl
AddModuleInfo module-name string

string module-name HTML :

Apache Module mod_isapi

<table>
<thead>
<tr>
<th>Description:</th>
<th>ISAPI Extensions within Apache for Windows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Base</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>isapi_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_isapi.c</td>
</tr>
<tr>
<td>Compatibility:</td>
<td>Win32 only</td>
</tr>
</tbody>
</table>

**Summary**

This module implements the Internet Server extension API. It allows Internet Server extensions (e.g. ISAPI .dll modules) to be served by Apache for Windows, subject to the noted restrictions.

ISAPI extension modules (.dll files) are written by third parties. The Apache Group does not author these modules, so we provide no support for them. Please contact the ISAPI's author directly if you are experiencing problems running their ISAPI extension. **Please do not post such problems to Apache's lists or bug reporting pages.**
In the server configuration file, use the `AddHandler` directive to associate ISAPI files with the `isapi-handler` handler, and map it to them with their file extensions. To enable any `.dll` file to be processed as an ISAPI extension, edit the `httpd.conf` file and add the following line:

```
AddHandler isapi-handler .dll
```

In older versions of the Apache server, `isapi-isa` was the proper handler name, rather than `isapi-handler`. As of 2.3 development versions of the Apache server, `isapi-isa` is no longer valid. You will need to change your configuration to use `isapi-handler` instead.

There is no capability within the Apache server to leave a requested module loaded. However, you may preload and keep a specific module loaded by using the following syntax in your `httpd.conf`:

```
ISAPICacheFile c:/WebWork/Scripts/ISAPI/mytest.dll
```

Whether or not you have preloaded an ISAPI extension, all ISAPI extensions are governed by the same permissions and restrictions as CGI scripts. That is, `Options ExecCGI` must be set for the directory that contains the ISAPI `.dll` file.

Review the `Additional Notes` and the `Programmer's Journal` for additional details and clarification of the specific ISAPI support offered by `mod_isapi`. 
Apache's ISAPI implementation conforms to all of the ISAPI 2.0 specification, except for some "Microsoft-specific" extensions dealing with asynchronous I/O. Apache's I/O model does not allow asynchronous reading and writing in a manner that the ISAPI could access. If an ISA tries to access unsupported features, including async I/O, a message is placed in the error log to help with debugging. Since these messages can become a flood, the directive ISAPILogNotSupported Off exists to quiet this noise.

Some servers, like Microsoft IIS, load the ISAPI extension into the server and keep it loaded until memory usage is too high, or unless configuration options are specified. Apache currently loads and unloads the ISAPI extension each time it is requested, unless the ISAPICacheFile directive is specified. This is inefficient, but Apache's memory model makes this the most effective method. Many ISAPI modules are subtly incompatible with the Apache server, and unloading these modules helps to ensure the stability of the server.

Also, remember that while Apache supports ISAPI Extensions, it does not support ISAPI Filters. Support for filters may be added at a later date, but no support is planned at this time.
If you are programming Apache 2.0 mod_isapi modules, you must limit your calls to ServerSupportFunction to the following directives:

**HSE_REQ_SEND_URL_REDIRECT_RESP**
Redirect the user to another location.
This must be a fully qualified URL (e.g. http://server/location).

**HSE_REQ_SEND_URL**
Redirect the user to another location.
This cannot be a fully qualified URL, you are not allowed to pass the protocol or a server name (e.g. simply /location). This redirection is handled by the server, not the browser.

**Warning**
In their recent documentation, Microsoft appears to have abandoned the distinction between the two HSE_REQ_SEND_URL functions. Apache continues to treat them as two distinct functions with different requirements and behaviors.

**HSE_REQ_SEND_RESPONSE_HEADER**
Apache accepts a response body following the header if it follows the blank line (two consecutive newlines) in the headers string argument. This body cannot contain NULLs, since the headers argument is NULL terminated.

**HSE_REQ_DONE_WITH_SESSION**
Apache considers this a no-op, since the session will be finished when the ISAPI returns from processing.

**HSE_REQ_MAP_URL_TO_PATH**
Apache will translate a virtual name to a physical name.
**HSE_APPEND_LOG_PARAMETER**

This logged message may be captured in any of the following logs:

- in the "\%{isapi-parameter}\n" component in a CustomLog directive
- in the \%q log component with the ISAPIAppendLogToQuery On directive
- in the error log with the ISAPIAppendLogToErrors On directive

The first option, the \%{isapi-parameter}\n component, is always available and preferred.

**HSE_REQ_IS_KEEP_CONN**

Will return the negotiated Keep-Alive status.

**HSE_REQ_SEND_RESPONSE_HEADER_EX**

Will behave as documented, although the fKeepConn flag is ignored.

**HSE_REQ_IS_CONNECTED**

Will report false if the request has been aborted.

Apache returns FALSE to any unsupported call to ServerSupportFunction, and sets the GetLastError value to ERROR_INVALID_PARAMETER.

ReadClient retrieves the request body exceeding the initial buffer (defined by ISAPIReadAheadBuffer). Based on the ISAPIReadAheadBuffer setting (number of bytes to buffer prior to calling the ISAPI handler) shorter requests are sent complete to the extension when it is invoked. If the request is longer, the ISAPI extension must use ReadClient to retrieve the remaining request body.
WriteClient is supported, but only with the HSE_IO_SYNC flag or no option flag (value of 0). Any other WriteClient request will be rejected with a return value of FALSE, and a GetLastError value of ERROR_INVALID_PARAMETER.

GetServerVariable is supported, although extended server variables do not exist (as defined by other servers.) All the usual Apache CGI environment variables are available from GetServerVariable, as well as the ALL_HTTP and ALL_RAW values.

Since httpd 2.0, mod_isapi supports additional features introduced in later versions of the ISAPI specification, as well as limited emulation of async I/O and the TransmitFile semantics. Apache httpd also supports preloading ISAPI .dlls for performance.
**Description:** Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the error log

**Syntax:** ISAPIAppendLogToErrors on|off

**Default:** ISAPIAppendLogToErrors off

**Context:** server config, virtual host, directory, .htaccess

**Override:** FileInfo

**Status:** Base

**Module:** mod_isapi

Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the server error log.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the query field</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>ISAPIAppendLogToQuery on</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>ISAPIAppendLogToQuery on</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td><strong>Override:</strong></td>
<td>FileInfo</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_isapi</td>
</tr>
</tbody>
</table>

Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the query field (appended to the CustomLog %q component).
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>ISAPI .dll files to be loaded at startup</th>
</tr>
</thead>
</table>
| **Syntax:**      | ISAPICacheFile *file-path* [*file-path]* *
|                  | ...                                    |
| **Context:**     | server config, virtual host            |
| **Status:**      | Base                                    |
| **Module:**      | mod_isapi                               |

Specifies a space-separated list of file names to be loaded when the Apache server is launched, and remain loaded until the server is shut down. This directive may be repeated for every ISAPI .dll file desired. The full path name of each file should be specified. If the path name is not absolute, it will be treated relative to ServerRoot.
**Description:** Fake asynchronous support for ISAPI callbacks

**Syntax:**  
ISAPIFakeAsync on|off

**Default:**  
ISAPIFakeAsync off

**Context:**  
server config, virtual host, directory, .htaccess

**Override:**  
FileInfo

**Status:**  
Base

**Module:**  
mod_isapi

While set to on, asynchronous support for ISAPI callbacks is simulated.
**Description:** Log unsupported feature requests from ISAPI extensions

**Syntax:** ISAPILogNotSupported on|off

**Default:** ISAPILogNotSupported off

**Context:** server config, virtual host, directory, .htaccess

**Override:** FileInfo

**Status:** Base

**Module:** mod_isapi

Logs all requests for unsupported features from ISAPI extensions in the server error log. This may help administrators to track down problems. Once set to on and all desired ISAPI modules are functioning, it should be set back to off.
**Description:** Size of the Read Ahead Buffer sent to ISAPI extensions

**Syntax:** ISAPIReadAheadBuffer size

**Default:** ISAPIReadAheadBuffer 49152

**Context:** server config, virtual host, directory, .htaccess

**Override:** FileInfo

**Status:** Base

**Module:** mod_isapi

Defines the maximum size of the Read Ahead Buffer sent to ISAPI extensions when they are initially invoked. All remaining data must be retrieved using the ReadClient callback; some ISAPI extensions may not support the ReadClient function. Refer questions to the ISAPI extension's author.
Apache Module mod_lbmethod_bybusyness

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Pending Request Counting load balancer scheduler algorithm for mod_proxy_balancer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>lbmethod_bybusyness_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_lbmethod_bybusyness.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Split off from mod_proxy_balancer in 2.3</td>
</tr>
</tbody>
</table>

**Summary**

This module does not provide any configuration directives of its own. It requires the services of mod_proxy_balancer, and provides the bybusyness load balancing method.

**Bugfix checklist**

httpd changelog  
Known issues  
Report a bug

**See also**

mod_proxy  
mod_proxy_balancer
Enabled via `lbmethod=bybusyness`, this scheduler keeps track of how many requests each worker is currently assigned at present. A new request is automatically assigned to the worker with the lowest number of active requests. This is useful in the case of workers that queue incoming requests independently of Apache, to ensure that queue length stays even and a request is always given to the worker most likely to service it the fastest and reduce latency.

In the case of multiple least-busy workers, the statistics (and weightings) used by the Request Counting method are used to break the tie. Over time, the distribution of work will come to resemble that characteristic of `byrequests` (as implemented by `mod_lbmethod_byrequests`).
Apache Module mod_lbmethod_byrequests

**Description:** Request Counting load balancer scheduler algorithm for `mod_proxy_balancer`

**Status:** Extension

**Module Identifier:** lbmethod_byrequests_module

**Source File:** mod_lbmethod_byrequests.c

**Compatibility:** Split off from `mod_proxy_balancer` in 2.3

**Summary**

This module does not provide any configuration directives of its own. It requires the services of `mod_proxy_balancer`, and provides the byrequests load balancing method.

**Bugfix checklist**

- [httpd changelog](http://httpd.apache.org/docs/2.4/changes.html)
- [Known issues](http://httpd.apache.org/docs/2.4/known.html)
- [Report a bug](http://httpd.apache.org/docs/2.4/bugzilla.html)

**See also**

- [mod_proxy](http://httpd.apache.org/docs/2.4/mod/mod_proxy.html)
- [mod_proxy_balancer](http://httpd.apache.org/docs/2.4/mod/mod_proxy_balancer.html)
Enabled via `lbmethod=byrequests`, the idea behind this scheduler is that we distribute the requests among the various workers to ensure that each gets their configured share of the number of requests. It works as follows:

`lbfactor` is how much we expect this worker to work, or the workers' work quota. This is a normalized value representing their "share" of the amount of work to be done.

`lbstatus` is how urgent this worker has to work to fulfill its quota of work.

The worker is a member of the load balancer, usually a remote host serving one of the supported protocols.

We distribute each worker's work quota to the worker, and then look which of them needs to work most urgently (biggest lbstatus). This worker is then selected for work, and its lbstatus reduced by the total work quota we distributed to all workers. Thus the sum of all lbstatus does not change(*) and we distribute the requests as desired.

If some workers are disabled, the others will still be scheduled correctly.

```plaintext
for each worker in workers
  worker lbstatus += worker lbfactor
  total factor += worker lbfactor
  if worker lbstatus > candidate lbstatus
    candidate = worker
  candidate lbstatus -= total factor
```

If a balancer is configured as follows:

```plaintext
worker a b c d
```
And $b$ gets disabled, the following schedule is produced:

<table>
<thead>
<tr>
<th>lbfactor</th>
<th>25</th>
<th>25</th>
<th>25</th>
<th>25</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbstatus</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

That is it schedules: $a \ c \ d \ a \ c \ d \ a \ c \ d \ ...$ Please note that:

<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbstatus</td>
<td>-50</td>
<td>0</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>lbstatus</td>
<td>-25</td>
<td>0</td>
<td>-25</td>
<td>50</td>
</tr>
<tr>
<td>lbstatus</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(repeat)

Has the exact same behavior as:

<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbfactor</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

This is because all values of $lbfactor$ are normalized with respect to the others. For:

<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbfactor</td>
<td>1</td>
<td>4</td>
<td>1</td>
</tr>
</tbody>
</table>

worker $b$ will, on average, get 4 times the requests that $a$ and $c$ will.

The following asymmetric configuration works as one would expect:

<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>70</td>
<td>30</td>
</tr>
<tr>
<td>lbfactor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>-30 30</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>40 -40</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>10 -10</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>-20 20</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>-50 50</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>20 -20</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>-10 10</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>-40 40</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>30 -30</td>
<td></td>
</tr>
<tr>
<td>lbstatus</td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

(repeat)

That is after 10 schedules, the schedule repeats and 7 a are selected with 3 b interspersed.
Apache Module mod_lbmethod_bytraffic

**Description:** Weighted Traffic Counting load balancer scheduler algorithm for mod_proxy_balancer

**Status:** Extension

**Module Identifier:** lbmethod_bytraffic_module

**Source File:** mod_lbmethod_bytraffic.c

**Compatibility:** Split off from mod_proxy_balancer in 2.3

**Summary**

This module does not provide any configuration directives of its own. It requires the services of mod_proxy_balancer, and provides the bytraffic load balancing method.

**Bugfix checklist**

httpd changelog
Known issues
Report a bug

**See also**

mod_proxy
mod_proxy_balancer
Enabled via `lbmethod=bytraffic`, the idea behind this scheduler is very similar to the Request Counting method, with the following changes:

`lbfactor` is *how much traffic, in bytes, we want this worker to handle*. This is also a normalized value representing their "share" of the amount of work to be done, but instead of simply counting the number of requests, we take into account the amount of traffic this worker has either seen or produced.

If a balancer is configured as follows:

```
worker    a  b  c
lbfactor  1  2  1
```

Then we mean that we want `b` to process twice the amount of bytes than `a` or `c` should. It does not necessarily mean that `b` would handle twice as many requests, but it would process twice the I/O. Thus, the size of the request and response are applied to the weighting and selection algorithm.

**Note:** Input and output bytes are weighted the same.
Apache Module mod_lbmethode_heartbeat

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Heartbeat Traffic Counting load balancer scheduler algorithm for mod_proxy_balancer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>lbmethod_heartbeat_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_lbmethode_heartbeat.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in version 2.3 and later</td>
</tr>
</tbody>
</table>

**Summary**

lbmethod=heartbeat uses the services of mod_heartmonitor to balance between origin servers that are providing heartbeat info via the mod_heartbeat module.

This modules load balancing algorithm favors servers with more ready (idle) capacity over time, but does not select the server with the most ready capacity every time. Servers that have 0 active clients are penalized, with the assumption that they are not fully initialized.

**Bugfix checklist**

[httpd changelog](httpd changelog)
[Known issues](Known issues)
[Report a bug](Report a bug)

**See also**

[mod_proxy](mod_proxy)
[mod_proxy_balancer](mod_proxy_balancer)
mod_heartbeat
mod_heartmonitor
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Path to read heartbeat data</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>HeartbeatStorage <em>file-path</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>HeartbeatStorage <em>logs/hb.dat</em></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_lbmethod_heartbeat</td>
</tr>
</tbody>
</table>

The **HeartbeatStorage** directive specifies the path to read heartbeat data. This flat-file is used only when **mod_slotmem_shm** is not loaded.
### Apache Module mod_ldap

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>LDAP connection pooling and result caching services for use by other LDAP modules</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>ldap_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>util_ldap.c</td>
</tr>
</tbody>
</table>

## Summary

This module was created to improve the performance of websites relying on backend connections to LDAP servers. In addition to the functions provided by the standard LDAP libraries, this module adds an LDAP connection pool and an LDAP shared memory cache.

To enable this module, LDAP support must be compiled into apr-util. This is achieved by adding the `--with-ldap` flag to the `configure` script when building Apache.

SSL/TLS support is dependent on which LDAP toolkit has been linked to APR. As of this writing, APR-util supports: OpenLDAP SDK (2.x or later), Novell LDAP SDK, Mozilla LDAP SDK, native Solaris LDAP SDK (Mozilla based) or the native Microsoft LDAP SDK. See the APR website for details.
The following is an example configuration that uses mod_ldap to increase the performance of HTTP Basic authentication provided by mod_authnz_ldap.

```plaintext
# Enable the LDAP connection pool and shared memory cache. Enable the LDAP cache status handler. Requires that mod_ldap and mod_authnz_ldap be loaded. Change the "yourdomain.example.com" to match your domain.

LDAPSharedCacheSize 500000
LDAPCacheEntries 1024
LDAPCacheTTL 600
LDAPOpCacheEntries 1024
LDAPOpCacheTTL 600

<Location "/ldap-status">
  SetHandler ldap-status

  Require host yourdomain.example.com

  Satisfy any
  AuthType Basic
  AuthName "LDAP Protected"
  AuthBasicProvider ldap
  AuthLDAPURL "ldap://127.0.0.1/dc=example"
  Require valid-user
</Location>
```
LDAP connections are pooled from request to request. This allows the LDAP server to remain connected and bound ready for the next request, without the need to unbind/connect/rebind. The performance advantages are similar to the effect of HTTP keepalives.

On a busy server it is possible that many requests will try and access the same LDAP server connection simultaneously. Where an LDAP connection is in use, Apache will create a new connection alongside the original one. This ensures that the connection pool does not become a bottleneck.

There is no need to manually enable connection pooling in the Apache configuration. Any module using this module for access to LDAP services will share the connection pool.

LDAP connections can keep track of the ldap client credentials used when binding to an LDAP server. These credentials can be provided to LDAP servers that do not allow anonymous binds during referral chasing. To control this feature, see the LDAPReferrals and LDAPReferralHopLimit directives. By default, this feature is enabled.
For improved performance, mod_ldap uses an aggressive caching strategy to minimize the number of times that the LDAP server must be contacted. Caching can easily double or triple the throughput of Apache when it is serving pages protected with mod_authnz_ldap. In addition, the load on the LDAP server will be significantly decreased.

mod_ldap supports two types of LDAP caching during the search/bind phase with a search/bind cache and during the compare phase with two operation caches. Each LDAP URL that is used by the server has its own set of these three caches.

**The Search/Bind Cache**

The process of doing a search and then a bind is the most time-consuming aspect of LDAP operation, especially if the directory is large. The search/bind cache is used to cache all searches that resulted in successful binds. Negative results (i.e., unsuccessful searches, or searches that did not result in a successful bind) are not cached. The rationale behind this decision is that connections with invalid credentials are only a tiny percentage of the total number of connections, so by not caching invalid credentials, the size of the cache is reduced.

mod_ldap stores the username, the DN retrieved, the password used to bind, and the time of the bind in the cache. Whenever a new connection is initiated with the same username, mod_ldap compares the password of the new connection with the password in the cache. If the passwords match, and if the cached entry is not too old, mod_ldap bypasses the search/bind phase.

The search and bind cache is controlled with the LDAPCacheEntries and LDAPCacheTTL directives.
Operation Caches

During attribute and distinguished name comparison functions, **mod_ldap** uses two operation caches to cache the compare operations. The first compare cache is used to cache the results of compares done to test for LDAP group membership. The second compare cache is used to cache the results of comparisons done between distinguished names.

Note that, when group membership is being checked, any sub-group comparison results are cached to speed future sub-group comparisons.

The behavior of both of these caches is controlled with the `LDAPOpCacheEntries` and `LDAPOpCacheTTL` directives.

Monitoring the Cache

**mod_ldap** has a content handler that allows administrators to monitor the cache performance. The name of the content handler is `ldap-status`, so the following directives could be used to access the **mod_ldap** cache information:

```
<Location "/server/cache-info">
    SetHandler ldap-status
</Location>
```

By fetching the URL `http://servername/cache-info`, the administrator can get a status report of every cache that is used by **mod_ldap** cache. Note that if Apache does not support shared memory, then each **httpd** instance has its own cache, so reloading the URL will result in different information each time, depending on which **httpd** instance processes the request.
The ability to create an SSL and TLS connections to an LDAP server is defined by the directives `LDAPTrustedGlobalCert`, `LDAPTrustedClientCert` and `LDAPTrustedMode`. These directives specify the CA and optional client certificates to be used, as well as the type of encryption to be used on the connection (none, SSL or TLS/STARTTLS).

```apache
# Establish an SSL LDAP connection on port 636. Requires that mod_ldap and mod_authnz_ldap be loaded. Change the "yourdomain.example.com" to match your domain.

LDAPTrustedGlobalCert CA_DER "/certs/certfile.der"

<Location "/ldap-status">
  SetHandler ldap-status

  Require host yourdomain.example.com

  Satisfy any
  AuthType Basic
  AuthName "LDAP Protected"
  AuthBasicProvider ldap
  AuthLDAPURL "ldaps://127.0.0.1/dc=example,dc=com?uid?one"
  Require valid-user
</Location>

# Establish a TLS LDAP connection on port 389. Requires that mod_ldap and mod_authnz_ldap be loaded. Change the "yourdomain.example.com" to match your domain.

LDAPTrustedGlobalCert CA_DER "/certs/certfile.der"

<Location "/ldap-status">
  SetHandler ldap-status
```
Require host yourdomain.example.com

Satisfy any
AuthType Basic
AuthName "LDAP Protected"
AuthBasicProvider ldap
AuthLDAPURL "ldap://127.0.0.1/dc=example"
Require valid-user
</Location>
The different LDAP SDKs have widely different methods of setting and handling both CA and client side certificates.

If you intend to use SSL or TLS, read this section CAREFULLY so as to understand the differences between configurations on the different LDAP toolkits supported.

**Netscape/Mozilla/iPlanet SDK**

CA certificates are specified within a file called cert7.db. The SDK will not talk to any LDAP server whose certificate was not signed by a CA specified in this file. If client certificates are required, an optional key3.db file may be specified with an optional password. The secmod file can be specified if required. These files are in the same format as used by the Netscape Communicator or Mozilla web browsers. The easiest way to obtain these files is to grab them from your browser installation.

Client certificates are specified per connection using the LDAPTrustedClientCert directive by referring to the certificate "nickname". An optional password may be specified to unlock the certificate's private key.

The SDK supports SSL only. An attempt to use STARTTLS will cause an error when an attempt is made to contact the LDAP server at runtime.

```bash
# Specify a Netscape CA certificate file
LDAPTrustedGlobalCert CA_CERT7_DB "'/certs/cert7.db"

# Specify an optional key3.db file for client
LDAPTrustedGlobalCert CERT_KEY3_DB "'/certs/key3.db"

# Specify the secmod file if required
LDAPTrustedGlobalCert CA_SECMOD "'/certs/secmod"

<Location "'/ldap-status">
    SetHandler ldap-status
</Location>
```
Novell SDK

One or more CA certificates must be specified for the Novell SDK to work correctly. These certificates can be specified as binary DER or Base64 (PEM) encoded files.

Note: Client certificates are specified globally rather than per connection, and so must be specified with the LDAPTrustedGlobalCert directive as below. Trying to set client certificates via the LDAPTrustedClientCert directive will cause an error to be logged when an attempt is made to connect to the LDAP server.

The SDK supports both SSL and STARTTLS, set using the LDAPTrustedMode parameter. If an ldaps:// URL is specified, SSL mode is forced, override this directive.

```bash
# Specify two CA certificate files
LDAPTrustedGlobalCert CA_DER "/certs/cacert1.der"
LDAPTrustedGlobalCert CA_BASE64 "/certs/cacert2.pem"

# Specify a client certificate file and key
LDAPTrustedGlobalCert CERT_BASE64 "/certs/cert1.pem"
LDAPTrustedGlobalCert KEY_BASE64 "/certs/key1.pem"
```
OpenLDAP SDK

One or more CA certificates must be specified for the OpenLDAP SDK to work correctly. These certificates can be specified as binary DER or Base64 (PEM) encoded files.

Both CA and client certificates may be specified globally (LDAPTrustedGlobalCert) or per-connection (LDAPTrustedClientCert). When any settings are specified per-connection, the global settings are superseded.

The documentation for the SDK claims to support both SSL and STARTTLS, however STARTTLS does not seem to work on all versions of the SDK. The SSL/TLS mode can be set using the LDAPTrustedMode parameter. If an ldaps:// URL is specified, SSL mode is forced. The OpenLDAP documentation notes that SSL (ldaps://) support has been deprecated to be replaced with TLS, although the SSL functionality still works.

# Specify two CA certificate files
LDAPTrustedGlobalCert CA_DER "'/certs/cacert1.der"
LDAPTrustedGlobalCert CA_BASE64 "'/certs/cacert2.pem"

<Location "'/ldap-status">
SetHandler ldap-status

  Require host youdomain.example.com

  LDAPTrustedClientCert CERT_BASE64 "'/certs/cert1.pem"
  LDAPTrustedClientCert KEY_BASE64 "'/certs/key1.pem"

  # CA certs respecified due to per-directory client certs
  LDAPTrustedClientCert CA_DER "'/certs/cacert1.der"
  LDAPTrustedClientCert CA_BASE64 "'/certs/cacert2.pem"
Satisfy any
AuthType Basic
AuthName "LDAP Protected"
AuthBasicProvider ldap
AuthLDAPURL "ldaps://127.0.0.1/dc=example,dc=com?uid?one"
Require valid-user
</Location>

**Solaris SDK**

SSL/TLS for the native Solaris LDAP libraries is not yet supported. If required, install and use the OpenLDAP libraries instead.

**Microsoft SDK**

SSL/TLS certificate configuration for the native Microsoft LDAP libraries is done inside the system registry, and no configuration directives are required.

Both SSL and TLS are supported by using the ldaps:// URL format, or by using the LDAPTrustedMode directive accordingly.

Note: The status of support for client certificates is not yet known for this toolkit.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Maximum number of entries in the primary LDAP cache</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>LDAPCacheEntries <em>number</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>LDAPCacheEntries 1024</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ldap</td>
</tr>
</tbody>
</table>

Specifies the maximum size of the primary LDAP cache. This cache contains successful search/binds. Set it to 0 to turn off search/bind caching. The default size is 1024 cached searches.
**Description:** Time that cached items remain valid

**Syntax:** LDAPCacheTTL  *seconds*

**Default:** LDAPCacheTTL  600

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

Specifies the time (in seconds) that an item in the search/bind cache remains valid. The default is 600 seconds (10 minutes).
**LDAPConnectionPoolTTL**

**Description:** Discard backend connections that have been sitting in the connection pool too long.

**Syntax:** `LDAPConnectionPoolTTL n`

**Default:** `LDAPConnectionPoolTTL -1`

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ldap

**Compatibility:** Apache HTTP Server 2.3.12 and later

Specifies the maximum age, in seconds, that a pooled LDAP connection can remain idle and still be available for use. Connections are cleaned up when they are next needed, not asynchronously.

A setting of 0 causes connections to never be saved in the backend connection pool. The default value of -1, and any other negative value, allows connections of any age to be reused.

For performance reasons, the reference time used by this directive is based on when the LDAP connection is returned to the pool, not the time of the last successful I/O with the LDAP server.

Since 2.4.10, new measures are in place to avoid the reference time from being inflated by cache hits or slow requests. First, the reference time is not updated if no backend LDAP connections were needed. Second, the reference time uses the time the HTTP request was received instead of the time the request is completed.

This timeout defaults to units of seconds, but accepts suffixes for milliseconds (ms), minutes (min), and hours (h).
LDAPConnectionTimeout

**Description:** Specifies the socket connection timeout in seconds

**Syntax:**

```plaintext
LDAPConnectionTimeout seconds
```

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

This directive configures the LDAP_OPT_NETWORK_TIMEOUT (or LDAP_OPT_CONNECT_TIMEOUT) option in the underlying LDAP client library, when available. This value typically controls how long the LDAP client library will wait for the TCP connection to the LDAP server to complete.

If a connection is not successful with the timeout period, either an error will be returned or the LDAP client library will attempt to connect to a secondary LDAP server if one is specified (via a space-separated list of hostnames in the AuthLDAPURL).

The default is 10 seconds, if the LDAP client library linked with the server supports the LDAP_OPT_NETWORK_TIMEOUT option.

```
LDAPConnectionTimeout is only available when the LDAP client library linked with the server supports the LDAP_OPT_NETWORK_TIMEOUT (or LDAP_OPT_CONNECT_TIMEOUT) option, and the ultimate behavior is dictated entirely by the LDAP client library.
```
**Description:** Enable debugging in the LDAP SDK

**Syntax:** LDAPLibraryDebug 7

**Default:** disabled

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

Turns on SDK-specific LDAP debug options that generally cause the LDAP SDK to log verbose trace information to the main Apache error log. The trace messages from the LDAP SDK provide gory details that can be useful during debugging of connectivity problems with backend LDAP servers.

This option is only configurable when Apache HTTP Server is linked with an LDAP SDK that implements LDAP_OPT_DEBUG or LDAP_OPT_DEBUG_LEVEL, such as OpenLDAP (a value of 7 is verbose) or Tivoli Directory Server (a value of 65535 is verbose).

The logged information will likely contain plaintext credentials being used or validated by LDAP authentication, so care should be taken in protecting and purging the error log when this directive is used.
**Description:** Number of entries used to cache LDAP compare operations

**Syntax:** `LDAP0pCacheEntries number`

**Default:** `LDAP0pCacheEntries 1024`

**Context:** server config

**Status:** Extension

**Module:** `mod_ldap`

This specifies the number of entries `mod_ldap` will use to cache LDAP compare operations. The default is 1024 entries. Setting it to 0 disables operation caching.
**Description:** Time that entries in the operation cache remain valid

**Syntax:** LDAPOpCacheTTL  *seconds*

**Default:** LDAPOpCacheTTL  600

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

Specifies the time (in seconds) that entries in the operation cache remain valid. The default is 600 seconds.
The maximum number of referral hops to chase before terminating an LDAP query.

SDK dependent, typically between 5 and 10

directory, .htaccess

AuthConfig

Extension

mod_ldap

This directive, if enabled by the LDAPReferrals directive, limits the number of referral hops that are followed before terminating an LDAP query.

Support for this tunable is uncommon in LDAP SDKs.
**Description:** Enable referral chasing during queries to the LDAP server.

**Syntax:** LDAPReferrals 0n|0ff|default

**Default:** LDAPReferrals 0n

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_ldap

**Compatibility:** The *default* parameter is available in Apache 2.4.7 and later

Some LDAP servers divide their directory among multiple domains and use referrals to direct a client when a domain boundary is crossed. This is similar to a HTTP redirect. LDAP client libraries may or may not chase referrals by default. This directive explicitly configures the referral chasing in the underlying SDK.

**LDAPReferrals** takes the following values:

**"on"**
When set to "on", the underlying SDK's referral chasing state is enabled, **LDAPReferralHopLimit** is used to override the SDK's hop limit, and an LDAP rebind callback is registered.

**"off"**
When set to "off", the underlying SDK's referral chasing state is disabled completely.

**"default"**
When set to "default", the underlying SDK's referral chasing state is not changed, **LDAPReferralHopLimit** is not used to override the SDK's hop limit, and no LDAP rebind callback is registered.
The directive **LDAPReferralHopLimit** works in conjunction with this directive to limit the number of referral hops to follow before terminating the LDAP query. When referral processing is enabled by a value of "On", client credentials will be provided, via a rebinding callback, for any LDAP server requiring them.
**Description:** Configures the number of LDAP server retries.

**Syntax:** LDAPRetries number-of-retries

**Default:** LDAPRetries 3

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

The server will retry failed LDAP requests up to `LDAPRetries` times. Setting this directive to 0 disables retries.

LDAP errors such as timeouts and refused connections are retryable.
**Description:** Configures the delay between LDAP server retries.

**Syntax:** `LDAPRetryDelay seconds`

**Default:** `LDAPRetryDelay 0`

**Context:** server config

**Status:** Extension

**Module:** `mod_ldap`

If `LDAPRetryDelay` is set to a non-zero value, the server will delay retrying an LDAP request for the specified amount of time. Setting this directive to 0 will result in any retry to occur without delay.

LDAP errors such as timeouts and refused connections are retryable.
**Description:** Sets the shared memory cache file

**Syntax:** LDAPSharedCacheFile `directory-path/filename`

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

Specifies the directory path and file name of the shared memory cache file. If not set, anonymous shared memory will be used if the platform supports it.
**Description:** Size in bytes of the shared-memory cache

**Syntax:** LDAPSharedCacheSize  bytes

**Default:** LDAPSharedCacheSize  500000

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

Specifies the number of bytes to allocate for the shared memory cache. The default is 500kb. If set to 0, shared memory caching will not be used and every HTTPD process will create its own cache.
**Description:** Specifies the timeout for LDAP search and bind operations, in seconds

**Syntax:** LDAPTimeout *seconds*

**Default:** LDAPTimeout 60

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

**Compatibility:** Apache HTTP Server 2.3.5 and later

This directive configures the timeout for bind and search operations, as well as the LDAP_OPT_TIMEOUT option in the underlying LDAP client library, when available.

If the timeout expires, httpd will retry in case an existing connection has been silently dropped by a firewall. However, performance will be much better if the firewall is configured to send TCP RST packets instead of silently dropping packets.

Timeouts for ldap compare operations requires an SDK with LDAP_OPT_TIMEOUT, such as OpenLDAP >= 2.4.4.
**Description:** Sets the file containing or nickname referring to a per connection client certificate. Not all LDAP toolkits support per connection client certificates.

**Syntax:** `LDAPTrustedClientCert type directory-path/filename/nickname [password]`

**Context:** directory, .htaccess

**Status:** Extension

**Module:** `mod_ldap`

It specifies the directory path, file name or nickname of a per connection client certificate used when establishing an SSL or TLS connection to an LDAP server. Different locations or directories may have their own independent client certificate settings. Some LDAP toolkits (notably Novell) do not support per connection client certificates, and will throw an error on LDAP server connection if you try to use this directive (Use the `LDAPTrustedGlobalCert` directive instead for Novell client certificates - See the SSL/TLS certificate guide above for details). The type specifies the kind of certificate parameter being set, depending on the LDAP toolkit being used. Supported types are:

- CA_DER - binary DER encoded CA certificate
- CA_BASE64 - PEM encoded CA certificate
- CERT_DER - binary DER encoded client certificate
- CERT_BASE64 - PEM encoded client certificate
- CERT_NICKNAME - Client certificate "nickname" (Netscape SDK)
- KEY_DER - binary DER encoded private key
- KEY_BASE64 - PEM encoded private key
**Description:** Sets the file or database containing global trusted Certificate Authority or global client certificates

**Syntax:** LDAPTrustedGlobalCert type directory-path/filename [password]

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

It specifies the directory path and file name of the trusted CA certificates and/or system wide client certificates [mod_ldap](#) should use when establishing an SSL or TLS connection to an LDAP server. Note that all certificate information specified using this directive is applied globally to the entire server installation. Some LDAP toolkits (notably Novell) require all client certificates to be set globally using this directive. Most other toolkits require clients certificates to be set per Directory or per Location using LDAPTrustedClientCert. If you get this wrong, an error may be logged when an attempt is made to contact the LDAP server, or the connection may silently fail (See the SSL/TLS certificate guide above for details). The type specifies the kind of certificate parameter being set, depending on the LDAP toolkit being used. Supported types are:

- **CA_DER** - binary DER encoded CA certificate
- **CA_BASE64** - PEM encoded CA certificate
- **CA_CERT7_DB** - Netscape cert7.db CA certificate database file
- **CA_SECMOD** - Netscape secmod database file
- **CERT_DER** - binary DER encoded client certificate
- **CERT_BASE64** - PEM encoded client certificate
- **CERT_KEY3_DB** - Netscape key3.db client certificate database file
- **CERT_NICKNAME** - Client certificate "nickname" (Netscape...
- CERT_PFX - PKCS#12 encoded client certificate (Novell SDK)
- KEY_DER - binary DER encoded private key
- KEY_BASE64 - PEM encoded private key
- KEY_PFX - PKCS#12 encoded private key (Novell SDK)
**Description:** Specifies the SSL/TLS mode to be used when connecting to an LDAP server.

**Syntax:** `LDAPTrustedMode` *type*

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ldap

The following modes are supported:

- NONE - no encryption
- SSL - ldaps:// encryption on default port 636
- TLS - STARTTLS encryption on default port 389

Not all LDAP toolkits support all the above modes. An error message will be logged at runtime if a mode is not supported, and the connection to the LDAP server will fail.

If an ldaps:// URL is specified, the mode becomes SSL and the setting of LDAPTrustedMode is ignored.
**LDAPVerifyServerCert Directive**

**Description:** Force server certificate verification

**Syntax:** LDAPVerifyServerCert *On|Off*

**Default:** LDAPVerifyServerCert *On*

**Context:** server config

**Status:** Extension

**Module:** mod_ldap

Specifies whether to force the verification of a server certificate when establishing an SSL connection to the LDAP server.
Apache mod_log_config

Bugfix checklist

httpd changelog
Known issues
Report a bug
### LogFormat CustomLog

```
" %"                             "%
```

<table>
<thead>
<tr>
<th>%</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%a</td>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>%A</td>
<td>IP</td>
<td></td>
</tr>
<tr>
<td>%B</td>
<td>HTTP</td>
<td></td>
</tr>
<tr>
<td>%b</td>
<td>HTTP CLF</td>
<td>1 0</td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Foobar}C</td>
<td>Foobar</td>
<td></td>
</tr>
<tr>
<td>%D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{FOOBAR}e</td>
<td>FOOBAR</td>
<td></td>
</tr>
<tr>
<td>%f</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%h</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Foobar}i</td>
<td>Foobar:</td>
<td></td>
</tr>
<tr>
<td>%l</td>
<td>(identd)</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IdentityCheck</td>
<td>On</td>
<td></td>
</tr>
<tr>
<td>%m</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Foobar}n</td>
<td>Foobar</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{Foobar}o</td>
<td>Foobar:</td>
<td></td>
</tr>
<tr>
<td>%p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>{format}p</td>
<td>local, remote</td>
<td></td>
</tr>
<tr>
<td>%P</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>%</td>
<td>ID</td>
<td></td>
</tr>
<tr>
<td>{format}P</td>
<td>hextid</td>
<td>APR 1.2.0</td>
</tr>
<tr>
<td>%q</td>
<td>( ? )</td>
<td></td>
</tr>
<tr>
<td>%r</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>%s</td>
<td>CLF ()</td>
<td></td>
</tr>
<tr>
<td>%t</td>
<td>format format</td>
<td></td>
</tr>
<tr>
<td>{format}t</td>
<td>strftime (3)</td>
<td></td>
</tr>
<tr>
<td>%T</td>
<td></td>
<td></td>
</tr>
<tr>
<td>%u</td>
<td>(( %s 401)</td>
<td></td>
</tr>
<tr>
<td>%U</td>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>%v</td>
<td>ServerName</td>
<td></td>
</tr>
<tr>
<td>%V</td>
<td>UseCanonicalName</td>
<td></td>
</tr>
<tr>
<td>%X</td>
<td>:</td>
<td></td>
</tr>
<tr>
<td></td>
<td>X =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>+ =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- =</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(Apache 1.3 %c</td>
<td></td>
</tr>
<tr>
<td>%I</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>%O</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

"%" HTTP
400 500 User-agent
"%!200,304,302{Referer}i" 3
Common Log Format (CLF)
"%h %l %u %t "%r" %>s %b"

Common Log Format
"%v %h %l %u %t "%r" %>s %b"

NCSA extended/combined
"%h %l %u %t "%r" %>s %b "%{Referer}i"
"%{User-agent}i"

Referer
"%{Referer}i -> %U"

Agent()
"%{User-agent}i"
| BufferLogs 0n|Off       |
|--------------------|
| BufferLogs Off     |

Base

mod_log_config

2.0.41

BufferedLogs

mod_log_config
CustomLog

file

ServerRoot

pipe

" |"

httpd

Unix

LogFormat

# CustomLog with format nickname
LogFormat "%h %l %u %t "%r" %>s %b" common
CustomLog logs/access_log common

# CustomLog with explicit format string
CustomLog logs/access_log "%h %l %u %t "%r" %>s %b"
mod_setenvif  mod_rewrite

SetEnvIf Request_URI \.gif$ gif-image
CustomLog gif-requests.log common env=gif-image
CustomLog nongif-requests.log common env=!gif-image

RefererIgnore

SetEnvIf Referer example\..com localreferer
CustomLog referer.log referer log referer env=!localreferer
GlobalLog

Sets filename and format of log file

GlobalLogfile|pipe format|nickname [env=[!]environment-variable| expr=expression]

Base
mod_log_config
Available in Apache HTTP Server 2.4.19 and later
LogFormat format|nickname [nickname]
LogFormat "%h %l %u %t "%r" %>s %b"

Base mod_log_config

LogFormat nickname

LogFormat format nickname
LogFormat CustomLog

nickname Nickname (vhost_common)
TransferLog file|pipe

Base
mod_log_config

Log Format

```
LogFormat "%h %l %u %t "%r" %>s %b "%{Referer}i" "%{User-agent}i"
TransferLog logs/access_log
```
Apache Module mod_log_debug

**Description:** Additional configurable debug logging

**Status:** Experimental

**Module Identifier:** log_debug_module

**Source File:** mod_log_debug.c

**Compatibility:** Available in Apache 2.3.14 and later
1. Log message after request to /foo/* is processed:

```xml
<Location "/foo/"
    LogMessage "/foo/ has been requested"
</Location>
```

2. Log message if request to /foo/* is processed in a sub-request:

```xml
<Location "/foo/"
    LogMessage "subrequest to /foo/" hook=
</Location>
```

The default log_transaction hook is not executed for sub-requests, therefore we have to use a different hook.

3. Log message if an IPv6 client causes a request timeout:

```xml
LogMessage "IPv6 timeout from %{REMOTE_ADDR}"
```

Note the placing of the double quotes for the expr= argument.

4. Log the value of the "X-Foo" request environment variable in each stage of the request:

```xml
<Location "/"
    LogMessage "%{reqenv:X-Foo}" hook=all
</Location>
```

Together with microsecond time stamps in the error log, hook=all also lets you determine the times spent in the
different parts of the request processing.
**Description:** Log user-defined message to error log

**Syntax:**
```
LogMessage message [hook=hook] [expr=expression]
```

**Default:** Unset

**Context:** directory

**Status:** Experimental

**Module:** mod_log_debug

This directive causes a user defined message to be logged to the error log. The message can use variables and functions from the `ap_expr` syntax. References to HTTP headers will not cause header names to be added to the Vary header. The messages are logged at loglevel info.

The hook specifies before which phase of request processing the message will be logged. The following hooks are supported:

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>translate_name</td>
</tr>
<tr>
<td>type_checker</td>
</tr>
<tr>
<td>quick_handler</td>
</tr>
<tr>
<td>map_to_storage</td>
</tr>
<tr>
<td>check_access</td>
</tr>
<tr>
<td>check_access_ex</td>
</tr>
<tr>
<td>insert_filter</td>
</tr>
<tr>
<td>check_authn</td>
</tr>
<tr>
<td>check_authz</td>
</tr>
<tr>
<td>fixups</td>
</tr>
<tr>
<td>handler</td>
</tr>
<tr>
<td>log_transaction</td>
</tr>
</tbody>
</table>
The default is `log_transaction`. The special value `all` is also supported, causing a message to be logged at each phase. Not all hooks are executed for every request.

The optional expression allows to restrict the message if a condition is met. The details of the expression syntax are described in the `ap_expr documentation`. References to HTTP headers will not cause the header names to be added to the Vary header.
Apache `mod_log_forensic`

forensic

Extension

log_forensic_module

mod_log_forensic.c

mod_unique_id

2.1

Bugfix checklist

- httpd changelog
- Known issues
- Report a bug

Apache
mod_log_config
ID forensic ID

mod_unique_id

forensic ID

( | )

+yQtJf8CoAB4AAFNXBIEAAAAA|GET /manual/de/images/down.gif
HTTP/1.1|Host:localhost%3a8080|User-Agent:Mozilla/5.0 (X11; U;
Linux i686; en-US; rv%3a1.6) Gecko/20040216
Firefox/0.8|Accept:image/png, etc...

ID:

-yQtJf8CoAB4AAFNXBIEAAAAA

check_forensic +/-
ForensicLog forensic
    mod_log_forensic forensic-id
id}n transfer

2

filename
    ServerRoot

pipe
    " |

httpd root root

Unix
Apache mod_logio

mod_log_config

Bugfix checklist
httpd changelog
Known issues
Report a bug

mod_log_config
Apache
I/O:
"%h %l %u %t "%r" %>s %b "\%{Referer}\i"
"\%{User-agent}\i" %I %O"
<table>
<thead>
<tr>
<th>Enable tracking of time to first byte (TTFB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>LogIOTrackTTFB ON</td>
</tr>
<tr>
<td>LogIOTrackTTFB OFF</td>
</tr>
<tr>
<td>. , . .htaccess</td>
</tr>
<tr>
<td>none</td>
</tr>
<tr>
<td>Extension</td>
</tr>
<tr>
<td>mod_logio</td>
</tr>
<tr>
<td>Apache HTTP Server 2.4.13 and later</td>
</tr>
</tbody>
</table>
Apache Module mod_lua

**Description:** Provides Lua hooks into various portions of the httpd request processing

**Status:** Experimental

**Module Identifier:** lua_module

**Source File:** mod_lua.c

**Compatibility:** 2.3 and later

**Summary**

This module allows the server to be extended with scripts written in the Lua programming language. The extension points (hooks) available with mod_lua include many of the hooks available to natively compiled Apache HTTP Server modules, such as mapping requests to files, generating dynamic responses, access control, authentication, and authorization.

More information on the Lua programming language can be found at the [Lua website](http://www.lua.org).

mod_lua is still in experimental state. Until it is declared stable, usage and behavior may change at any time, even between stable releases of the 2.4.x series. Be sure to check the CHANGES file before upgrading.

**Warning**

This module holds a great deal of power over httpd, which is both a strength and a potential security risk. It is **not** recommended that you use this module on a server that is shared with users you do not trust, as it can be abused to change the internal workings of httpd.
The basic module loading directive is

```
LoadModule lua_module modules/mod_lua.so
```

mod_lua provides a handler named lua-script, which can be used with a `SetHandler` or `AddHandler` directive:

```
<Files "*.lua">
  SetHandler lua-script
</Files>
```

This will cause mod_lua to handle requests for files ending in .lua by invoking that file's handle function.

For more flexibility, see `LuaMapHandler`. 
In the Apache HTTP Server API, the handler is a specific kind of hook responsible for generating the response. Examples of modules that include a handler are `mod_proxy`, `mod_cgi`, and `mod_status`.

`mod_lua` always looks to invoke a Lua function for the handler, rather than just evaluating a script body CGI style. A handler function looks something like this:

```lua
-- example.handler
require "string"

--[[
This is the default method name for Lua handlers, see the function-name in the LuaMapHandler directive for entry point.
--]]
function handle(r)
  r.content_type = "text/plain"

  if r.method == 'GET' then
    r:puts("Hello Lua World!
"
            for k, v in pairs(r:parseargs()) do
              r:puts(string.format("%s: %s
", k, v)
          end
    elseif r.method == 'POST' then
      r:puts("Hello Lua World!
"
            for k, v in pairs(r:parsebody()) do
              r:puts(string.format("%s: %s
", k, v)
          end
    elseif r.method == 'PUT' then
      -- use our own Error contents
      r:puts("Unsupported HTTP method ").
```
This handler function just prints out the uri or form encoded arguments to a plaintext page.

This means (and in fact encourages) that you can have multiple handlers (or hooks, or filters) in the same script.
mod_authz_core provides a high-level interface to authorization that is much easier to use than using into the relevant hooks directly. The first argument to the Require directive gives the name of the responsible authorization provider. For any Require line, mod_authz_core will call the authorization provider of the given name, passing the rest of the line as parameters. The provider will then check authorization and pass the result as return value.

The authz provider is normally called before authentication. If it needs to know the authenticated user name (or if the user will be authenticated at all), the provider must return apache2.AUTHZ_DENIED_NO_USER. This will cause authentication to proceed and the authz provider to be called a second time.

The following authz provider function takes two arguments, one ip address and one user name. It will allow access from the given ip address without authentication, or if the authenticated user matches the second argument:

```lua
authz_provider.lua

require 'apache2'

function authz_check_foo(r, ip, user)
    if r.useragent_ip == ip then
        return apache2.AUTHZ_GRANTED
    elseif r.user == nil then
        return apache2.AUTHZ_DENIED_NO_USER
    elseif r.user == user then
        return apache2.AUTHZ_GRANTED
    else
```
The following configuration registers this function as provider foo and configures it for URL /:

```
LuaAuthzProvider foo authz_provider.lua auth
<Location "/”>
    Require foo 10.1.2.3 john_doe
</Location>
```
Hook functions are how modules (and Lua scripts) participate in the processing of requests. Each type of hook exposed by the server exists for a specific purpose, such as mapping requests to the file system, performing access control, or setting mime types:

<table>
<thead>
<tr>
<th>Hook phase</th>
<th>mod_lua directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quick handler</td>
<td>\texttt{LuaQuickHandler}</td>
<td>This is the first hook that will be called after a request has been mapped to a host or virtual host</td>
</tr>
<tr>
<td>Translate name</td>
<td>\texttt{LuaHookTranslateName}</td>
<td>This phase translates the requested URI into a filename on the system. Modules such as \texttt{mod_alias} and \texttt{mod_rewrite} operate in this phase.</td>
</tr>
<tr>
<td>Map to storage</td>
<td>\texttt{LuaHookMapToStorage}</td>
<td>This phase maps files to their physical, cached or external/proxied storage. It can be used by proxy or caching modules</td>
</tr>
<tr>
<td>Check Access</td>
<td>\texttt{LuaHookAccessChecker}</td>
<td>This phase checks whether a client has access to a resource. This phase is run before the user is authenticated, so beware.</td>
</tr>
<tr>
<td>Phase</td>
<td>LuaHook Function</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Check User ID</td>
<td>LuaHookCheckUserID</td>
<td>This phase it used to check the negotiated user ID</td>
</tr>
<tr>
<td>Check Authorization</td>
<td>LuaHookAuthChecker or LuaAuthzProvider</td>
<td>This phase authorizes a user based on the negotiated credentials, such as user ID, client certificate etc.</td>
</tr>
<tr>
<td>Check Type</td>
<td>LuaHookTypeChecker</td>
<td>This phase checks the requested file and assigns a content type and a handler to it</td>
</tr>
<tr>
<td>Fixups</td>
<td>LuaHookFixups</td>
<td>This is the final &quot;fix anything&quot; phase before the content handlers are run. Any last-minute changes to the request should be made here.</td>
</tr>
<tr>
<td>Content handler</td>
<td>fx. .lua files or through LuaMapHandler</td>
<td>This is where the content is handled. Files are read, parsed, some are run, and the result is sent to the client</td>
</tr>
<tr>
<td>Logging</td>
<td>LuaHookLog</td>
<td>Once a request has been handled, it enters several logging phases, which logs the request in either the error or access log. ModLua is able to hook into the start of</td>
</tr>
</tbody>
</table>
Hook functions are passed the request object as their only argument (except for LuaAuthzProvider, which also gets passed the arguments from the Require directive). They can return any value, depending on the hook, but most commonly they'll return OK, DONE, or DECLINED, which you can write in Lua as `apache2.OK`, `apache2.DONE`, or `apache2.DECLINED`, or else an HTTP status code.

```
translate_name.lua
-- example hook that rewrites the URI to another
require 'apache2'

function translate_name(r)
   if r.uri == '/translate-name' then
      r.filename = r.document_root .. '/find_me.txt'
      return apache2.OK
   end
   -- we don't care about this URL, give another module a chance
   return apache2.DECLINED
end
```

```
translate_name2.lua
-- example hook that rewrites one URI to another;
-- returns apache2.DECLINED to give other URL mappers
-- a chance to work on the substitution, including the core translate_name
-- hook which maps based on the DocumentRoot.

Note: Use the early/late flags in the directive
or after mod_alias.
```
require 'apache2'

function translate_name(r)
    if r.uri == "translate-name" then
        r.uri = "find_me.txt"
        return apache2.DECLINED
    end
    return apache2.DECLINED
end

**request_rec**

The request_rec is mapped in as a userdata. It has a metatable which lets you do useful things with it. For the most part it has the same fields as the request_rec struct, many of which are writable as well as readable. (The table fields' content can be changed, but the fields themselves cannot be set to different tables.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Lua type</th>
<th>Writable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowoverrides</td>
<td>string</td>
<td>no</td>
<td>The AllowOverride options applied to the current request.</td>
</tr>
<tr>
<td>ap_auth_type</td>
<td>string</td>
<td>no</td>
<td>If an authentication check was made, this is set to the type authentication (f.x. basic).</td>
</tr>
<tr>
<td>args</td>
<td>string</td>
<td>yes</td>
<td>The query string arguments extracted from the request foo=bar&amp;name=johnsmith.</td>
</tr>
<tr>
<td>assbackwards</td>
<td>boolean</td>
<td>no</td>
<td>Set to true if this is an HTTP/0.9 style request (no headers).</td>
</tr>
<tr>
<td>auth_name</td>
<td>string</td>
<td>no</td>
<td>The realm name used for authentication (if applicable).</td>
</tr>
<tr>
<td>banner</td>
<td>string</td>
<td>no</td>
<td>The server banner, f.x. HTTP Server/2.4.3 openssl/0.9.8c.</td>
</tr>
<tr>
<td>basic_auth_pw</td>
<td>string</td>
<td>no</td>
<td>The basic auth password sent with this request, if any.</td>
</tr>
<tr>
<td>canonical_filename</td>
<td>string</td>
<td>no</td>
<td>The canonical filename of the request.</td>
</tr>
<tr>
<td>content_encoding</td>
<td>string</td>
<td>no</td>
<td>The content encoding of the request.</td>
</tr>
<tr>
<td>Property</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>content_type</td>
<td>string</td>
<td>yes</td>
<td>The content type of the current request, as determined in the type_check phase (f.x. image/gif)</td>
</tr>
<tr>
<td>context_prefix</td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>context_document_root</td>
<td>string</td>
<td>no</td>
<td></td>
</tr>
<tr>
<td>document_root</td>
<td>string</td>
<td>no</td>
<td>The document root of the host</td>
</tr>
<tr>
<td>err_headers_out</td>
<td>table</td>
<td>no</td>
<td>MIME header environment for the response, printed even on errors and persist across internal redirects</td>
</tr>
<tr>
<td>filename</td>
<td>string</td>
<td>yes</td>
<td>The file name that the request maps to, f.x. /www/example.com/foo.txt. This can be changed in the translate-name or map-to-storage phases of a request to allow the handler (or script handlers) to serve a different file than was requested.</td>
</tr>
<tr>
<td>handler</td>
<td>string</td>
<td>yes</td>
<td>The name of the handler that should serve this request, f.x. lua-script by mod_lua. For example, the AddHandler directives, built into mod_lua to allow a handler to serve up a file that would otherwise not be served by it.</td>
</tr>
<tr>
<td>headers_in</td>
<td>table</td>
<td>yes</td>
<td>MIME header environment from</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Default</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>headers_out</td>
<td>table</td>
<td>yes</td>
<td>MIME headers table for the response.</td>
</tr>
<tr>
<td>hostname</td>
<td>string</td>
<td>no</td>
<td>The host name, as set by the Host: header or by a full URI.</td>
</tr>
<tr>
<td>is_https</td>
<td>boolean</td>
<td>no</td>
<td>Whether or not this request is done via HTTPS.</td>
</tr>
<tr>
<td>is_initial_req</td>
<td>boolean</td>
<td>no</td>
<td>Whether this request is the initial request or a sub-request.</td>
</tr>
<tr>
<td>limit_req_body</td>
<td>number</td>
<td>no</td>
<td>The size limit of the request body for this request.</td>
</tr>
<tr>
<td>log_id</td>
<td>string</td>
<td>no</td>
<td>The ID to identify request in access and error log.</td>
</tr>
<tr>
<td>method</td>
<td>string</td>
<td>no</td>
<td>The request method, e.g., POST.</td>
</tr>
<tr>
<td>notes</td>
<td>table</td>
<td>yes</td>
<td>A list of notes that can be passed on from one module to another.</td>
</tr>
<tr>
<td>options</td>
<td>string</td>
<td>no</td>
<td>The Options directive applied to the current request.</td>
</tr>
<tr>
<td>path_info</td>
<td>string</td>
<td>no</td>
<td>The PATH_INFO extracted from this request.</td>
</tr>
<tr>
<td>port</td>
<td>number</td>
<td>no</td>
<td>The server port used by the request.</td>
</tr>
<tr>
<td>protocol</td>
<td>string</td>
<td>no</td>
<td>The protocol used, e.g., HTTP.</td>
</tr>
</tbody>
</table>
| proxyreq          | string  | yes     | Denotes whether this request or no request is a proxy request. Generally set in post_read_request phase of a request.
<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>string</td>
<td>no</td>
<td>The contents of the header.</td>
</tr>
<tr>
<td>remaining</td>
<td>number</td>
<td>no</td>
<td>The number of bytes remaining to be read from the request body.</td>
</tr>
<tr>
<td>server_built</td>
<td>string</td>
<td>no</td>
<td>The time the server executable was built.</td>
</tr>
<tr>
<td>server_name</td>
<td>string</td>
<td>no</td>
<td>The server name for this request.</td>
</tr>
<tr>
<td>some_auth_required</td>
<td>boolean</td>
<td>no</td>
<td>Whether some authorization is/was required for this request.</td>
</tr>
<tr>
<td>subprocess_env</td>
<td>table</td>
<td>yes</td>
<td>The environment variables set for this request.</td>
</tr>
<tr>
<td>started</td>
<td>number</td>
<td>no</td>
<td>The time the server was (re)started, in seconds since the epoch (Jan 1, 1970).</td>
</tr>
<tr>
<td>status</td>
<td>number</td>
<td>yes</td>
<td>The (current) HTTP return code for this request, f.x. GET /foo/bar HTTP/1.1.</td>
</tr>
<tr>
<td>the_request</td>
<td>string</td>
<td>no</td>
<td>The request string as sent by the client, f.x. GET /foo/bar HTTP/1.1.</td>
</tr>
<tr>
<td>unparsed_uri</td>
<td>string</td>
<td>no</td>
<td>The unparsed URI of the request.</td>
</tr>
<tr>
<td>uri</td>
<td>string</td>
<td>yes</td>
<td>The URI after it has been parsed by httpd.</td>
</tr>
<tr>
<td>user</td>
<td>string</td>
<td>yes</td>
<td>If an authentication check has been made, this is set to the name of the authenticated user.</td>
</tr>
<tr>
<td>useragent_ip</td>
<td>string</td>
<td>no</td>
<td>The IP of the user agent making the request.</td>
</tr>
</tbody>
</table>
The request_rec object has (at least) the following methods:

```lua
r:flush() -- flushes the output buffer.
        -- Returns true if the flush was successful, false otherwise.
while we_have_stuff_to_send do
    r:puts("Bla bla bla\n") -- print something
    r:flush() -- flush the buffer (send to client)
    r.usleep(500000) -- fake processing time
end

r:addoutputfilter(name|function) -- add an output filter:
r:addoutputfilter("fooFilter") -- add the fooFilter to the output stream

r:sendfile(filename) -- sends an entire file:
if use_sendfile_thing then
    r:sendfile("/var/www/large_file.img")
end

r:parseargs() -- returns two tables; one standard key/value table for regular GET data,
                -- and one for multi-value data (e.g., foo=1&foo=2&foo=3):
local GET, GETMULTI = r:parseargs()
r:puts("Your name is: " .. GET['name'] or "Unknown")

r:parsebody([sizeLimit]) -- parse the request body as a POST and return two lua tables,
-- just like r:parseargs.
-- An optional number may be passed to specify the maximum number
```
-- of bytes to parse.

local POST, POSTMULTI = r:parsebody(1024*1024)

r:puts("Your name is: " .. POST['name'] or "Unknown")

r:puts("hello", " world", "!") -- print to response body, self explanatory

r:write("a single string") -- print to response body, self explanatory

r:escape_html("<html>test</html>") -- Escapes HTML code and returns the escaped result

r:base64_encode(string) -- Encodes a string

local encoded = r:base64_encode("This is a test") -- returns VGhpcyBpcyBhIHRlc3Q=

r:base64_decode(string) -- Decodes a Base64 encoded string

local decoded = r:base64_decode("VGhpcyBpcyBhIHRlc3Q=") -- returns 'This is a test'

r:md5(string) -- Calculates and returns the MD5 digest of a string (binary safe)

local hash = r:md5("This is a test") -- returns ce114e4501d2f4e2dcea3e17b546f339

r:sha1(string) -- Calculates and returns the SHA1 digest of a string (binary safe)

local hash = r:sha1("This is a test") -- returns a54d88e06612d820bc3be72877c74f257b561b19
r:escape(string) -- URL-Escapes a string:

```lua
local url = "http://foo.bar/1 2 3 & 4 + 5"
local escaped = r:escape(url) -- returns 'http%3a%2f%2ffoo.bar%2f1%202%203%20%26%204%20%2b%205'
```

r:unescape(string) -- Unescapes an URL-escaped string:

```lua
local url = "http%3a%2f%2ffoo.bar%2f1%2b2%2b3%26%2b4%2b5"
local unescaped = r:unescape(url) -- returns 'http://foo.bar/1+2+3&+4+5'
```

r:construct_url(string) -- Constructs an URL:

```lua
local url = r:construct_url(r.uri)
```

r.mpm_query(number) -- Queries the server for MPM information using ap_mpm_query:

```lua
local mpm = r.mpm_query(14)
if mpm == 1 then
    r:puts("This server uses the Event MPM")
end
```

r:expr(string) -- Evaluates an expr string.

```lua
if r:expr("%{HTTP_HOST} =~ /^www/") then
    r:puts("This host name starts with www")
end
```

r:scoreboard_process(a) -- Queries the server for...
local process = r:scoreboard_process(1)
r:puts("Server 1 has PID " .. process.pid)

r:scoreboard_worker(a, b) -- Queries for information about the worker thread.
local thread = r:scoreboard_worker(1, 1)
r:puts("Server 1's thread 1 has thread ID " .. thread.tid .. " and is in " .. thread.status .. " status")

r:clock() -- Returns the current time with microsecond precision

r:requestbody(filename) -- Reads and returns the request body of a request.
-- If 'filename' is specified, it instead saves the contents to that file:
local input = r:requestbody()
r:puts("You sent the following request body:
"
)
r:puts(input)

r:add_input_filter(filter_name) -- Adds 'filter_name' as an input filter

r.module_info(module_name) -- Queries the server for information about a module
local mod = r.module_info("mod_lua.c")
if mod then
    for k, v in pairs(mod.commands) do
        r:puts( "\%%s: \%%s\n\":format(k,v)) -- print out all directives accepted by this module
    end
end
r:loaded_modules() -- Returns a list of modules loaded by httpd:
for k, module in pairs(r:loaded_modules()) do
  r:puts("I have loaded module " .. module .. 
end

r:runtime_dir_relative(filename) -- Compute the name of a run-time file (e.g., shared memory "file") -- relative to the appropriate run-time directory.

r:server_info() -- Returns a table containing server information, such as the name of the httpd executable etc.

r:set_document_root(file_path) -- Sets the document root for the request to file_path

r:set_context_info(prefix, docroot) -- Sets the context prefix and context document root for a request

r:os_escape_path(file_path) -- Converts an OS path to a URL in an OS dependent way

r:escape_logitem(string) -- Escapes a string for logging

r.strcmp_match(string, pattern) -- Checks if 'string' matches 'pattern' using strcmp_match (globs).
-- fx. whether 'www.example.com' matches '*.example.com':
local match = r.strcmp_match("foobar.com", "foo*.com")
if match then
  r:puts("foobar.com matches foo*.com")
end
r:set_keepalive() -- Sets the keepalive status for a request. Returns true if possible, false otherwise.

r:make_etag() -- Constructs and returns the etag for the current request.

r:send_interim_response(clear) -- Sends an interim (1xx) response to the client.
-- if 'clear' is true, available headers will be sent and cleared.

r:custom_response(status_code, string) -- Constructs and sets a custom response for a given status code.
-- This works much like the ErrorDocument directive:

r:custom_response(404, "Baleted!")

r:exists_config_define(string) -- Checks whether a configuration definition exists or not:
if r:exists_config_define("FOO") then
  r:puts("httpd was probably run with -DFOO,
or it was defined in the configuration")
end

r:state_query(string) -- Queries the server for state information

r:stat(filename [,wanted]) -- Runs stat() on a file and returns a table with file information:
local info = r:stat("/var/www/foo.txt")
if info then
  r:puts("This file exists and was last modified at: 	"	.. info.modified")
end
r:regex(string, pattern [,flags]) -- Runs a regular expression match on a string, returning captures if matched:

local matches = r:regex("foo bar baz", [[foo \((\w+)\)\ ([\S]*)\]])
if matches then
    r:puts("The regex matched, and the last word captured \(\2\) was:\ ") .. matches[2]
end

-- Example ignoring case sensitivity:
local matches = r:regex("FOO bar BAz", [[(foo)\ bar\ BAz\]])

-- Flags can be a bitwise combination of:
-- 0x01: Ignore case
-- 0x02: Multiline search

r.usleep(number_of_microseconds) -- Puts the script to sleep for a given number of microseconds.

r:dbacquire(dbType[, dbParams]) -- Acquires a connection to a database and returns a database class.
-- See 'Database connectivity'

r:ivm_set("key", value) -- Set an Inter-VM variable.
-- These values persist
-- and so should only
-- be used if MaxConnectionsPerChild > 0
-- Values can be numbers,
-- strings and booleans,
-- and are stored on a per process basis

r:ivm_get("key") -- Fetches a variable set by ivm_set.
-- It exists or nil if no such variable exists.

-- An example getter/setter that saves a global variable
function handle(r)
    -- First VM to call this will get no value
    local foo = r:ivm_get("cached_data")
end
if not foo then
  foo = do_some_calcs() -- fake some
  r:ivm_set("cached_data", foo) -- set it globally
end
r:puts("Cached data is: ", foo)
end

r:htpassword(string [,algorithm [,cost]]) -- Creates a password hash from a string.
  -- algorithm: 0 = APMD5 (default), 1 = SHA, 2 = BCRYPT, 3 = CRYPT.
  -- cost: only valid with BCRYPT algorithm (default = 5).

r:mkdir(dir [,mode]) -- Creates a directory and sets mode to optional mode parameter.

r:mkrmdir(dir [,mode]) -- Creates directories recursive and sets mode to optional mode parameter.

r:rmdir(dir) -- Removes a directory.

r:touch(file [,mtime]) -- Sets the file modification time to current time or to optional mtime msec value.

r:get_direntries(dir) -- Returns a table with all directory entries.

function handle(r)
  local dir = r.context_document_root
  for _, f in ipairs(r:get_direntries(dir)) do
    local info = r:stat(dir .. "/" .. f)
    if info then
      local mtime = os.date(fmt, info.mtime / 1000000)
      local ftype = (info.filetype == 2) and "[dir]" or "[file]"
      r:puts( ("%s %s %10i %s\n"):format(ftype, mtime, info.size, f))
    end
  end
end
r.date_parse_rfc(string) -- Parses a date/time string and returns seconds since epoch.

r:getcookie(key) -- Gets a HTTP cookie

r:setcookie{
    key = [key],
    value = [value],
    expires = [expiry],
    secure = [boolean],
    httponly = [boolean],
    path = [path],
    domain = [domain]
} -- Sets a HTTP cookie, for instance:

r:setcookie{
    key = "cookie1",
    value = "HDHfa9eyffh396rt",
    expires = os.time() + 86400,
    secure = true
}

r:wsupgrade() -- Upgrades a connection to WebSockets if possible and requested:
if r:wsupgrade() then -- if we can upgrade:
    r:wswrite("Welcome to websockets!") -- write something to the client
    r:wsclose() -- goodbye!
end
r:wsread() -- Reads a WebSocket frame from a WebSocket upgraded connection (see above):
local line, isFinal = r:wsread() -- isFinal denotes whether this is the final frame.
r:wswrite("You wrote: " .. line)

r:wswrite(line) -- Writes a frame to a WebSocket client:
r:wswrite("Hello, world!")

r:wsclose() -- Closes a WebSocket request and terminates it for httpd:
if r:wsupgrade() then
    r:wswrite("Write something: ")
    local line = r:wsread() or "nothing"
    r:wswrite("You wrote: " .. line);
    r:wswrite("Goodbye!")
    r:wsclose()
end
Logging Functions

-- examples of logging messages
r:trace1("This is a trace log message")

r:debug("This is a debug log message")

r:info("This is an info log message")

r:notice("This is a notice log message")

r:warn("This is a warn log message")

r:err("This is an err log message")

r:alert("This is an alert log message")

r:crit("This is a crit log message")

r:emerg("This is an emerg log message")
A package named apache2 is available with (at least) the following contents.

**apache2.OK**
- internal constant OK. Handlers should return this if they've handled the request.

**apache2.DECLINED**
- internal constant DECLINED. Handlers should return this if they are not going to handle the request.

**apache2.DONE**
- internal constant DONE.

**apache2.version**
- Apache HTTP server version string

**apache2.HTTP_MOVED_TEMPORARILY**
- HTTP status code

**apache2.PROXYREQ_NONE, apache2.PROXYREQ_PROXY, apache2.PROXYREQ_REVERSE, apache2.PROXYREQ_RESPONSE**
- internal constants used by mod_proxy

**apache2.AUTHZ_DENIED, apache2.AUTHZ_GRANTED, apache2.AUTHZ_NEUTRAL, apache2.AUTHZGENERAL_ERROR, apache2.AUTHZ_DENIED_NO_USER**
- internal constants used by mod_authz_core

(Other HTTP status codes are not yet implemented.)
Filter functions implemented via LuaInputFilter or LuaOutputFilter are designed as three-stage non-blocking functions using coroutines to suspend and resume a function as buckets are sent down the filter chain. The core structure of such a function is:

```
function filter(r)
  -- Our first yield is to signal that we are ready to receive buckets.
  -- Before this yield, we can set up our environment, check for conditions,
  -- and, if we deem it necessary, decline filtering a request altogether:
  if something_bad then
    return -- This would skip this filter.
  end
  -- Regardless of whether we have data to prepend, a yield MUST be called there.
  -- Note that only output filters can prepend data. Input filters must use the
  -- final stage to append data to the content:
  coroutine.yield([optional header to be prepended to the content])

  -- After we have yielded, buckets will be sent to us, one by one, and we can
  -- do whatever we want with them and then pass on the result.
  -- Buckets are stored in the global variable 'bucket', so we create a loop
  -- that checks if 'bucket' is not nil:
  while bucket ~= nil do
    local output = mangle(bucket) -- Do some stuff to the content
    coroutine.yield(output) -- Return our new content to the filter chain
  end

  -- Once the buckets are gone, 'bucket' is set to nil, which will exit the
  -- loop and land us here. Anything extra we want to append to the content
  -- can be done by doing a final yield here:
  coroutine.yield([optional footer to be appended to the content])
end
```
Mod_lua implements a simple database feature for querying and running commands on the most popular database engines (mySQL, PostgreSQL, FreeTDS, ODBC, SQLite, Oracle) as well as mod_dbd.

The example below shows how to acquire a database handle and return information from a table:

```lua
function handle(r)
    -- Acquire a database handle
    local database, err = r:dbacquire("mysql")
    if not err then
        -- Select some information from it
        local results, err = database:select(r, "SELECT `name`, `age` FROM `people` WHERE 1")
        if not err then
            local rows = results(0) -- fetch all rows synchronously
            for k, row in pairs(rows) do
                r:puts( string.format("Name: \%s, Age: \%s<br/>", row[1], row[2])
            end
        else
            r:puts("Database query error: \%s", err)
        end
    else
        r:puts("Could not connect to the database: \%s", err)
    end
end
```

To utilize mod_dbd, specify mod_dbd as the database type, or leave the field blank:

```lua
local database = r:dbacquire("mod_dbd")
```
Database object and contained functions

The database object returned by dbacquire has the following methods:

Normal select and query from a database:

```
-- Run a statement and return the number of local affected, errmsg = database:query(r, "DELETE FROM `tbl` WHERE 1")
-- Run a statement and return a result set
local result, errmsg = database:select(r, "SELECT * FROM `people` WHERE 1")
```

Using prepared statements (recommended):

```
-- Create and run a prepared statement:
local statement, errmsg = database:prepare(r, "DELETE FROM `tbl` WHERE `age` > %u")
if not errmsg then
    local result, errmsg = statement:query(20) -- run the statement with age > 20
end

-- Fetch a prepared statement from a DBDPrepareSQL directive:
local statement, errmsg = database:prepared(r, "someTag")
if not errmsg then
    local result, errmsg = statement:select("John Doe", 123) -- inject the values "John Doe" and 123 into the statement
end
```

Escaping values, closing databases etc:

```
-- Escape a value for use in a statement:
local escaped = database:escape(r, [["'|blabla"]])

-- Close a database connection and free up handles:
database:close()
```
-- Check whether a database connection is up
local connected = database:active()

Working with result sets

The result set returned by `db:select` or by the prepared statement functions created through `db:prepare` can be used to fetch rows synchronously or asynchronously, depending on the row number specified:

- `result(0)` fetches all rows in a synchronous manner, returning a table of rows.
- `result(-1)` fetches the next available row in the set, asynchronously.
- `result(N)` fetches row number `N`, asynchronously:

```plaintext
-- fetch a result set using a regular query
local result, err = db:select(r, "SELECT * FROM `tbl` WHERE 1")

local rows = result(0) -- Fetch ALL rows synchronously.
local row = result(-1) -- Fetch the next available row asynchronously.
local row = result(1234) -- Fetch row number 1234, asynchronously.
local row = result(-1, true) -- Fetch the next available row, using row names as key indexes.
```

One can construct a function that returns an iterative function to iterate over all rows in a synchronous or asynchronous way, depending on the `async` argument:

```plaintext
function rows(resultset, async)
    local a = 0
    local function getnext()
        a = a + 1
        local row = resultset(-1)
    end

    if async then
        getnext
    else
        local rows = resultset(0)
        for i, row in ipairs(rows) do
            print(row)
        end
    end
end
```
return row and a or nil, row
end
if not async then
    return pairs(resultset(0))
else
    return getnext, self
end
end

local statement, err = db:prepare(r, "SELECT * FROM `tbl` WHERE `age` > %u")
if not err then
    -- fetch rows asynchronously:
    local result, err = statement:select(20)
    if not err then
        for index, row in rows(result, true) do
            ....
        end
    end
end

    -- fetch rows synchronously:
    local result, err = statement:select(20)
    if not err then
        for index, row in rows(result, false) do
            ....
        end
    end
end

Closing a database connection

Database handles should be closed using database:close() when they are no longer needed. If you do not close them manually, they will eventually be garbage collected and closed by mod_lua, but you may end up having too many unused connections to the database if you leave the closing up to mod_lua. Essentially, the following two measures are the same:
Precautions when working with databases

Although the standard query and run functions are freely available, it is recommended that you use prepared statements whenever possible, to both optimize performance (if your db handle lives on for a long time) and to minimize the risk of SQL injection attacks. run and query should only be used when there are no variables inserted into a statement (a static statement). When using dynamic statements, use db:prepare or db:prepared.
**Description:** Plug an authorization provider function into `mod_authz_core`  
**Syntax:** LuaAuthzProvider provider_name/path/to/lua/script.lua function_name  
**Context:** server config  
**Status:** Experimental  
**Module:** mod_lua  
**Compatibility:** 2.4.3 and later

After a lua function has been registered as authorization provider, it can be used with the `Require` directive:

```
LuRoot "/usr/local/apache2/lua"
LuaAuthzProvider foo authz.lua authz_check_foo
<Location "/">
    Require foo johndoe
</Location>
```

```lua
require "apache2"
function authz_check_foo(r, who)
    if r.user ~= who then return apache2.AUTHZ_DENIED
    return apache2.AUTHZ_GRANTED
end
```
**Description:** Configure the compiled code cache.

**Syntax:**
```
LuaCodeCache stat | forever | never
```

**Default:**
```
LuaCodeCache stat
```

**Context:**
```
server config, virtual host, directory, .htaccess
```

**Override:**
```
All
```

**Status:** Experimental

**Module:** mod_lua

Specify the behavior of the in-memory code cache. The default is stat, which stats the top level script (not any included ones) each time that file is needed, and reloads it if the modified time indicates it is newer than the one it has already loaded. The other values cause it to keep the file cached forever (don't stat and replace) or to never cache the file.

In general stat or forever is good for production, and stat or never for development.

**Examples:**
```
LuaCodeCache stat
LuaCodeCache forever
LuaCodeCache never
```
Description: Provide a hook for the access_checker phase of request processing

Syntax: LuaHookAccessChecker
/path/to/lua/script.lua
hook_function_name [early|late]

Context: server config, virtual host, directory, .htaccess

Override: All

Status: Experimental

Module: mod_lua

Compatibility: The optional third argument is supported in 2.3.15 and later

Add your hook to the access_checker phase. An access checker hook function usually returns OK, DECLINED, or HTTP_FORBIDDEN.

Ordering
The optional arguments "early" or "late" control when this script runs relative to other modules.
**Description:** Provide a hook for the auth_checker phase of request processing

**Syntax:**
```
LuaHookAuthChecker
/path/to/lua/script.lua
hook_function_name [early|late]
```

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

**Compatibility:** The optional third argument is supported in 2.3.15 and later

Invoke a lua function in the auth_checker phase of processing a request. This can be used to implement arbitrary authentication and authorization checking. A very simple example:

```
require 'apache2'

-- fake authcheck hook
-- If request has no auth info, set the res
-- return a 401 to ask the browser for basic
-- If request has auth info, don't actually
-- pretend we got userid 'foo' and validate
-- Then check if the userid is 'foo' and accept
function authcheck_hook(r)
    -- look for auth info
    auth = r.headers_in['Authorization']
    if auth ~= nil then
        -- fake the user
        r.user = 'foo'
    end
end
```
if r.user == nil then
    r:debug("authcheck: user is nil, retu
    r.err_headers_out['WWW-Authenticate']
    return 401
elseif r.user == "foo" then
    r:debug('user foo: OK')
else
    r:debug("authcheck: user='" .. r.user
    r.err_headers_out['WWW-Authenticate']
    return 401
end
return apache2.OK
end

**Ordering**

The optional arguments "early" or "late" control when this script runs relative to other modules.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Provide a hook for the check_user_id phase of request processing</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>LuaHookCheckUserID /path/to/lua/script.lua hook_function_name [early</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td><strong>Override:</strong></td>
<td>All</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_lua</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>The optional third argument is supported in 2.3.15 and later</td>
</tr>
</tbody>
</table>

### Ordering
The optional arguments "early" or "late" control when this script runs relative to other modules.
**Description:** Provide a hook for the fixups phase of a request processing

**Syntax:** LuaHookFixups /path/to/lua/script.lua

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

Just like LuaHookTranslateName, but executed at the fixups phase
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Provide a hook for the insert_filter phase of request processing</th>
</tr>
</thead>
</table>
| **Syntax:**      | LuaHookInsertFilter  
<path/to/lua/script.lua>  
hook_function_name |
| **Context:**     | server config, virtual host, directory, .htaccess |
| **Override:**    | All |
| **Status:**      | Experimental |
| **Module:**      | mod_lua |

Not Yet Implemented
**Description:** Provide a hook for the access log phase of a request processing

**Syntax:**
```
LuaHookLog /path/to/lua/script.lua log_function_name
```

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

This simple logging hook allows you to run a function when httpd enters the logging phase of a request. With it, you can append data to your own logs, manipulate data before the regular log is written, or prevent a log entry from being created. To prevent the usual logging from happening, simply return `apache2.DONE` in your logging handler, otherwise return `apache2.OK` to tell httpd to log as normal.

Example:

```
LuaHookLog "/path/to/script.lua" logger
```

```lua
-- /path/to/script.lua --
function logger(r)
  -- flip a coin:
  -- If 1, then we write to our own Lua log in the main log.
  -- If 2, then we just sanitize the output log the sanitized bits.
  if math.random(1,2) == 1 then
    -- Log stuff ourselves and don't log
    local f = io.open("/foo/secret.log"
    if f then
```
Something secret happened.

```
f:write("Something secret happened.

f:close()
end
```

```
return apache2.DONE -- Tell httpd not
else
r.uri = r.uri:gsub("somesecretstuff",
return apache2.OK -- tell httpd to
end
```

```
end
```
**Description:** Provide a hook for the map_to_storage phase of request processing

**Syntax:**
```
LuaHookMapToStorage
/path/to/lua/script.lua
hook_function_name
```

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

Like [LuaHookTranslateName](#) but executed at the map-to-storage phase of a request. Modules like mod_cache run at this phase, which makes for an interesting example on what to do here:

```latex
require"apache2"
cached_files = {}

function read_file(filename)
    local input = io.open(filename, "r")
    if input then
        local data = input:read("*a")
        cached_files[filename] = data
        file = cached_files[filename]
        input:close()
    end
    return cached_files[filename]
end

function check_cache(r)
```

if r.filename:match("%.png$") then -- Only match PNG files
    local file = cached_files[r.filename]
    if not file then
        file = read_file(r.filename) -- Read file into cache
    end
    if file then -- If file exists, write it out
        r.status = 200
        r:write(file)
        r:info("Sent %s to client from cache")
    end
end
return apache2.DONE -- skip default handler for PNG files
end
return apache2.DECLINED -- If we had nothing to do, let others serve this.
**Description:** Provide a hook for the translate name phase of request processing

**Syntax:**
```
LuaHookTranslateName /path/to/lua/script.lua hook_function_name [early|late]
```

**Context:** server config, virtual host

**Override:** All

**Status:** Experimental

**Module:** mod_lua

**Compatibility:** The optional third argument is supported in 2.3.15 and later

Add a hook (at APR_HOOK_MIDDLE) to the translate name phase of request processing. The hook function receives a single argument, the request_rec, and should return a status code, which is either an HTTP error code, or the constants defined in the apache2 module: apache2.OK, apache2.DECLINED, or apache2.DONE.

For those new to hooks, basically each hook will be invoked until one of them returns apache2.OK. If your hook doesn't want to do the translation it should just return apache2.DECLINED. If the request should stop processing, then return apache2.DONE.

Example:
```
# httpd.conf
LuaHookTranslateName "/scripts/conf/hooks.lua"
```

```
-- /scripts/conf/hooks.lua --
require "apache2"
function silly_mapper(r)
```
if r.uri == "/" then
    r.filename = "/var/www/home.lua"
    return apache2.OK
else
    return apache2.DECLINED
end
end

**Context**

This directive is not valid in `<Directory>`, `<Files>`, or `htaccess` context.

**Ordering**

The optional arguments "early" or "late" control when this script runs relative to other modules.
**Description:** Provide a hook for the type_checker phase of request processing

**Syntax:**
```
LuaHookTypeChecker
/path/to/lua/script.lua
hook_function_name
```

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

This directive provides a hook for the type_checker phase of the request processing. This phase is where requests are assigned a content type and a handler, and thus can be used to modify the type and handler based on input:

```lua
function type_checker(r)
  if r.uri:match("%.to_gif$") then
    r.content_type = "image/gif"
    r.handler = "gifWizard"
    r.filename = r.uri:gsub("%.to_gif$", "")
  end
  return apache2.OK
end

return apache2.DECLINED
end
```
**Description:** Controls how parent configuration sections are merged into children

**Syntax:** LuaInherit none|parent-first|parent-last

**Default:** LuaInherit parent-first

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

**Compatibility:** 2.4.0 and later

By default, if LuaHook* directives are used in overlapping Directory or Location configuration sections, the scripts defined in the more specific section are run after those defined in the more generic section (LuaInherit parent-first). You can reverse this order, or make the parent context not apply at all.

In previous 2.3.x releases, the default was effectively to ignore LuaHook* directives from parent configuration sections.
**Description:** Provide a Lua function for content input filtering

**Syntax:**

```
LuaInputFilter filter_name
/path/to/lua/script.lua
function_name
```

**Context:** server config

**Status:** Experimental

**Module:** mod_lua

**Compatibility:** 2.4.5 and later

Provides a means of adding a Lua function as an input filter. As with output filters, input filters work as coroutines, first yielding before buffers are sent, then yielding whenever a bucket needs to be passed down the chain, and finally (optionally) yielding anything that needs to be appended to the input data. The global variable bucket holds the buckets as they are passed onto the Lua script:

```
LuaInputFilter myInputFilter "/www/filter.lua"
<Files "*.lua">
    SetInputFilter myInputFilter
</Files>
```
The input filter supports denying/skipping a filter if it is deemed unwanted:

```lua
function input_filter(r)
    if not good then
        return -- Simply deny filtering, pass on the original content instead
    end
    coroutine.yield() -- wait for buckets
    ... -- insert filter stuff here
end
```

See "Modifying contents with Lua filters" for more information.
**Description:** Map a path to a lua handler

**Syntax:**

```
LuaMapHandler uri-pattern
/path/to/lua/script.lua [function-name]
```

**Context:**
server config, virtual host, directory, .htaccess

**Override:**
All

**Status:** Experimental

**Module:** mod_lua

This directive matches a uri pattern to invoke a specific handler function in a specific file. It uses PCRE regular expressions to match the uri, and supports interpolating match groups into both the file path and the function name. Be careful writing your regular expressions to avoid security issues.

**Examples:**

```
LuaMapHandler "/(\w+)/(/\w+)" "/scripts/$1.lua" "handle_$2"
```

This would match uri's such as /photos/show?id=9 to the file /scripts/photos.lua and invoke the handler function handle_show on the lua vm after loading that file.

```
LuaMapHandler "/bingo" "/scripts/wombat.lua"
```

This would invoke the "handle" function, which is the default if no specific function name is provided.
**Description:** Provide a Lua function for content output filtering

**Syntax:**
```
LuaOutputFilter filter_name
/path/to/lua/script.lua
function_name
```

**Context:** server config

**Status:** Experimental

**Module:** mod_lua

**Compatibility:** 2.4.5 and later

Provides a means of adding a Lua function as an output filter. As with input filters, output filters work as coroutines, first yielding before buffers are sent, then yielding whenever a bucket needs to be passed down the chain, and finally (optionally) yielding anything that needs to be appended to the input data. The global variable bucket holds the buckets as they are passed onto the Lua script:

```
LuaOutputFilter myOutputFilter "/www/filter.lua"
<Files "]*.lua"
SetOutputFilter myOutputFilter
</Files>
```

```lua
--[[
 Example output filter that escapes all HTML entities
]]--
function output_filter(r)
coroutine.yield("(Handled by myOutputFilter)<br/>
") -- Prepend some data to the output, yield and wait for buckets.
while bucket do -- For each bucket, do...
    local output = r:escape_html(bucket)
coroutine.yield(output) -- Send converted data down the chain
end
```
As with the input filter, the output filter supports denying/skipping a filter if it is deemed unwanted:

```lua
function output_filter(r)
    if not r.content_type:match("text/html")
        return -- Simply deny filtering, pass on the original content instead
    end
    coroutine.yield() -- wait for buckets
    ... -- insert filter stuff here
end
```

**Lua filters with mod_filter**

When a Lua filter is used as the underlying provider via the `FilterProvider` directive, filtering will only work when the `filter-name` is identical to the `provider-name`.

See "[Modifying contents with Lua filters](#)" for more information.
**Description:** Add a directory to lua's package.cpath

**Syntax:** LuaPackageCPath

/path/to/include/?.soa

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

Add a path to lua's shared library search path. Follows the same conventions as lua. This just munges the package.cpath in the lua vms.
**Description:** Add a directory to lua's package.path

**Syntax:** LuaPackagePath /path/to/include/?.lua

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

Add a path to lua's module search path. Follows the same conventions as lua. This just munges the package.path in the lua vms.

**Examples:**

```
LuaPackagePath "./scripts/lib/?.lua"
LuaPackagePath "./scripts/lib/?/init.lua"
```
lua-quick-handler

**Description:**
Provide a hook for the quick handler of request processing

**Syntax:**
LuaQuickHandler /path/to/script.lua
hook_function_name

**Context:**
server config, virtual host

**Override:**
All

**Status:**
Experimental

**Module:**
mod_lua

This phase is run immediately after the request has been mapped to a virtual host, and can be used to either do some request processing before the other phases kick in, or to serve a request without the need to translate, map to storage et cetera. As this phase is run before anything else, directives such as `<Location>` or `<Directory>` are void in this phase, just as URIs have not been properly parsed yet.

**Context**
This directive is not valid in `<Directory>`, `<Files>`, or htaccess context.
**Description:** Specify the base path for resolving relative paths for mod_lua directives

**Syntax:** LuaRoot /path/to/a/directory

**Context:** server config, virtual host, directory, .htaccess

**Override:** All

**Status:** Experimental

**Module:** mod_lua

Specify the base path which will be used to evaluate all relative paths within mod_lua. If not specified they will be resolved relative to the current working directory, which may not always work well for a server.
<table>
<thead>
<tr>
<th>Description:</th>
<th>One of once, request, conn, thread -- default is once</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax:</td>
<td>LuaScope oncescencene</td>
</tr>
<tr>
<td>Default:</td>
<td>LuaScope once</td>
</tr>
<tr>
<td>Context:</td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td>Override:</td>
<td>All</td>
</tr>
<tr>
<td>Status:</td>
<td>Experimental</td>
</tr>
<tr>
<td>Module:</td>
<td>mod_lua</td>
</tr>
</tbody>
</table>

Specify the life cycle scope of the Lua interpreter which will be used by handlers in this "Directory." The default is "once"

- **once:**
  - use the interpreter once and throw it away.

- **request:**
  - use the interpreter to handle anything based on the same file within this request, which is also request scoped.

- **conn:**
  - Same as request but attached to the connection_rec

- **thread:**
  - Use the interpreter for the lifetime of the thread handling the request (only available with threaded MPMs).

- **server:**
  - This one is different than others because the server scope is quite long lived, and multiple threads will have the same server_rec. To accommodate this, server scoped Lua states are stored in an apr resource list. The min and max arguments specify the minimum and maximum number of Lua states to keep in the pool.
Generally speaking, the thread and server scopes execute roughly 2-3 times faster than the rest, because they don't have to spawn new Lua states on every request (especially with the event MPM, as even keepalive requests will use a new thread for each request). If you are satisfied that your scripts will not have problems reusing a state, then the thread or server scopes should be used for maximum performance. While the thread scope will provide the fastest responses, the server scope will use less memory, as states are pooled, allowing f.x. 1000 threads to share only 100 Lua states, thus using only 10% of the memory required by the thread scope.
Apache Module mod_macro

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Provides macros within apache httpd runtime configuration files</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>macro_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_macro.c</td>
</tr>
</tbody>
</table>

**Summary**

Provides macros within Apache httpd runtime configuration files, to ease the process of creating numerous similar configuration blocks. When the server starts up, the macros are expanded using the provided parameters, and the result is processed as along with the rest of the configuration file.
Macros are defined using `<Macro>` blocks, which contain the portion of your configuration that needs to be repeated, complete with variables for those parts that will need to be substituted.

For example, you might use a macro to define a `<VirtualHost>` block, in order to define multiple similar virtual hosts:

```
<Macro VHost $name $domain>
<VirtualHost *:80>
    ServerName $domain
    ServerAlias www.$domain
    DocumentRoot "/var/www/vhosts/$name"
    ErrorLog "/var/log/httpd/$name.error_log"
    CustomLog "/var/log/httpd/$name.access_log" combined
</VirtualHost>
</Macro>
```

Macro names are case-insensitive, like httpd configuration directives. However, variable names are case sensitive.

You would then invoke this macro several times to create virtual hosts:

```
Use VHost example example.com
Use VHost myhost hostname.org
Use VHost apache apache.org

UndefMacro VHost
```

At server startup time, each of these `Use` invocations would be expanded into a full virtualhost, as described by the `<Macro>` definition.
The **UndefMacro** directive is used so that later macros using the same variable names don't result in conflicting definitions.

A more elaborate version of this example may be seen below in the Examples section.
Parameter names should begin with a sigil such as $, %, or @, so that they are clearly identifiable, and also in order to help deal with interactions with other directives, such as the core Define directive. Failure to do so will result in a warning. Nevertheless, you are encouraged to have a good knowledge of your entire server configuration in order to avoid reusing the same variables in different scopes, which can cause confusion.

Parameters prefixed with either $ or % are not escaped. Parameters prefixes with @ are escaped in quotes.

Avoid using a parameter which contains another parameter as a prefix, (For example, $win and $winter) as this may cause confusion at expression evaluation time. In the event of such confusion, the longest possible parameter name is used.

If you want to use a value within another string, it is useful to surround the parameter in braces, to avoid confusion:

```
<Macro DocRoot ${docroot}>
    DocumentRoot "/var/www/${docroot}/htdocs"
</Macro>
```
Virtual Host Definition

A common usage of `mod_macro` is for the creation of dynamically-generated virtual hosts.

```xml
## Define a VHost Macro for repetitive configurations

<Macro VHost $host $port $dir>
    Listen $port
    <VirtualHost *:$port>
        ServerName $host
        DocumentRoot "$dir"

        # Public document root
        <Directory "$dir">
            Require all granted
        </Directory>

        # limit access to intranet subdir.
        <Directory "$dir/intranet">
            Require ip 10.0.0.0/8
        </Directory>
    </VirtualHost>
</Macro>

## Use of VHost with different arguments.

Use VHost www.apache.org 80 /vhosts/apache/htdocs
Use VHost example.org 8080 /vhosts/example/htdocs
Use VHost www.example.fr 1234 /vhosts/example/htdocs

Removal of a macro definition
```
It's recommended that you undefine a macro once you've used it. This avoids confusion in a complex configuration file where there may be conflicts in variable names.

```xml
<Macro DirGroup $dir $group>
    <Directory "$dir">
        Require group $group
    </Directory>
</Macro>

Use DirGroup /www/apache/private private
Use DirGroup /www/apache/server admin

UndefMacro DirGroup
```
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Define a configuration file macro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td><code>&lt;Macro name [par1 .. parN]&gt; ...&lt;/Macro&gt;</code></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_macro</td>
</tr>
</tbody>
</table>

The `<Macro>` directive controls the definition of a macro within the server runtime configuration files. The first argument is the name of the macro. Other arguments are parameters to the macro. It is good practice to prefix parameter names with any of `${%@}`, and not macro names with such characters.

```
<Macro LocalAccessPolicy>
    Require ip 10.2.16.0/24
</Macro>

<Macro RestrictedAccessPolicy $ipnumbers>
    Require ip $ipnumbers
</Macro>
```
**Description:** Undefine a macro

**Syntax:**

```
UndefMacro name
```

**Context:**

server config, virtual host, directory

**Status:**

Base

**Module:**

mod_macro

The **UndefMacro** directive undefines a macro which has been defined beforehand.

```
UndefMacro LocalAccessPolicy
UndefMacro RestrictedAccessPolicy
```
The **Use** directive controls the use of a macro. The specified macro is expanded. It must be given the same number of arguments as in the macro definition. The provided values are associated to their corresponding initial parameters and are substituted before processing.

```
Use LocalAccessPolicy
...
Use RestrictedAccessPolicy "192.54.172.0/24
...
```

is equivalent, with the macros defined above, to:

```
Require ip 10.2.16.0/24
...
Require ip 192.54.172.0/24 192.54.148.0/24
```
**Apache mod_mime**

- AddCharset
- AddEncoding
- AddHandler
- AddLanguage
- AddType
- encoding, content-language, MIME (content-type)
- TypesConfig
- MIME

- mod_mime
- AddInputFilter
- mod_negotiation Multiviews

- SetOutputFilter
- mod_mime
- Last-Modified
- 'touch' ()
Bugfix checklist

httpd changelog
Known issues
Report a bug

MimeMagicFile
AddDefaultCharset
ForceType
DefaultType
SetHandler
SetInputFilter
SetOutputFilter
Configure handler based on final extension only

<FilesMatch \.cgi$>
    SetHandler cgi-script
</FilesMatch>
MIME
UUencoding

HTTP/1.1 RFC 14.11

Content-Encoding

( )

Microsoft Word .zip pkzip

Apache

Content-encoding: pkzip
mime

MIME

AddType ( MimeMagicFile
AddInputFilter, AddOutputFilter

Charset

Apache Content-Language

Content-Language: en, fr
Content-Type: text/plain; charset=ISO-8859-1

charset
AddCharset charset extension [extension] ...

.,.,.htaccess

FileInfo

mod_mime

AddCharset charset

MIME charset

AddLanguage ja .ja
AddCharset EUC-JP .euc
AddCharset ISO-2022-JP .jis
AddCharset SHIFT_JIS .jis

(xxxx.jis.ja) AddCharset charset

extension

- mod_negotiation
- AddDefaultCharset
AddEncoding MIME-enc extension [extension]
...
,,.htaccess
FileInfo
mod_mime

AddEncoding extension

AddEncoding x-gzip .gz
AddEncoding x-compress .Z

.gz  x-gzip
x-zip  x-compress
x-Apache
compress          deflate
extension
AddHandler handler-name extension [extension]

...,

,.htaccess

FileInfo

mod_mime

extension handler-name
".cgi" CGI

AddHandler cgi-script .cgi

httpd.conf 
".cgi" CGI

extension

- SetHandler
AddInputFilter  

filter [...;filter...] extension [extension] ...

, ., .htaccess

FileInfo

mod_mime

2.0.26

AddInputFilter  

extension  POST

SetInputFilter

filter

extension

RemoveInputFilter

SetInputFilter
AddLanguage MIME-lang extension [extension]  
  ...
  , , .htaccess
  FileInfo
  :
  mod_mime

AddLanguage en .en
AddLanguage fr .fr

xxxx.en.Z compress (language

AddLanguage en .en
AddLanguage en-gb .en
AddLanguage en-us .en

.en en-us

extension

• mod_negotiation
AddOutputFilter filter[;filter...] extension
   [extension] ...

.,,.htaccess
FileInfo

mod_mime
2.0.26

AddOutputFilter extension
   AddOutputFilterByType
   .shtml SSI
e mod_deflate

AddOutputFilter INCLUDES;DEFLATE shtml

extension

- RemoveOutputFilter
- SetOutputFilter
AddType MIME-type extension [extension] ...
AddType . . . , .htaccess
AddType FileInfo
AddType mod_mime

AddType extension MIME ( 
AddType image/gif .gif

Example
AddType image/jpeg jpeg jpg jpe

MIME TypesConfig Add

extension

- DefaultType
- ForceType
DefaultLanguage Apache

(AddLanguage .fr .de)

DefaultLanguage

AddLanguage

DefaultLanguage en

- mod_negotiation
ModMimeUsePathInfo: path_info

ModMimeUsePathInfo	On|Off
ModMimeUsePathInfo	Off

mod_mime:

Apache	2.0.41

ModMimeUsePathInfo: Off

ModMimeUsePathInfo: On

/bar (foo.shtml)
/bar/foo.shtml
AddOutputFileter INCLUDES .shtml

ModMimeUsePathInfo: On

- AcceptPathInfo
MultiviewsMatch

MultiviewsMatch

Any|NegotiatedOnly|Filters|Handlers
[Handlers|Filters]

MultiviewsMatch NegotiatedOnly

,,.htaccess

FileInfo

mod_mime

2.0.26

MultiviewsMatch mod_negotiation Multiviews:
Multiviews (index.html)
index.html.fr index.html.gz)

NegotiatedOnly


Multi

500 index.html.cgi 1000 index.html.pl
.cgi .asis .asi

mod_mime A

.bak

Multiviews

MultiviewsMatch Handlers Filters

- Options
- mod_negotiation
RemoveCharset

extension

RemoveCharset .html .shtml
RemoveEncoding

/foof/.htaccess:
AddEncoding x-gzip .gz
AddType text/plain .asc
<Files *.*.asc>
  RemoveEncoding .gz
</Files>

foo.gz gzip  
foo.gz.asc

RemoveEncoding  AddEncoding

extension
RemoveHandler

/foo/.htaccess:
AddHandler server-parsed .html

/foo/bar/.htaccess:
RemoveHandler .html

/foo/bar  .html SSI( mod_include)

extension
RemoveInputFilter

extension

- AddInputFilter
- SetInputFilter

mod_mime

2.0.26
RemoveLanguage extension [extension] ...
, .htaccess
FileInfo
mod_mime
2.0.24

RemoveLanguage

extension
RemoveOutputFilter

extension

FileInfo

mod_mime

2.0.26

RemoveOutputFilter

shtml

- AddOutputFilter
<table>
<thead>
<tr>
<th><strong>RemoveType</strong></th>
<th><strong>extension</strong></th>
<th><strong>[extension]</strong></th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>, . . . . . . . .</td>
<td>, , , . . . . .</td>
<td>. . . . . . . .</td>
<td>. . . . . . . .</td>
</tr>
<tr>
<td>FileInfo</td>
<td>FileInfo</td>
<td>FileInfo</td>
<td>FileInfo</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>mod_mime</td>
<td>mod_mime</td>
<td>mod_mime</td>
<td>mod_mime</td>
</tr>
</tbody>
</table>

**foo/.htaccess:**

RemoveType .cgi

/ foo/ . cgi

RemoveType  

AddType  

extension

MIME

Default
TypesConfig

```
  mime.types
  TypesConfig file-path
  TypesConfig conf/mime.types
  :
  :
  :
  :
  : mod_mime
```

TypesConfig  MIME   file

IANA

http://www.iana.org/assignments/media-types/index.html

AddType

AddType

MIME-type [extension] ...

(('#')

(1) IANA (2) Server Project  category/x-subtype

- `mod_mime_magic`

Copyright 2017 The Apache Software Foundation.  
Apache License, Version 2.0.
Apache Module mod_mime_magic

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

**Status:** Extension

**Module Identifier:** mime_magic_module

**Source File:** mod_mime_magic.c

**Summary**

This module determines the [MIME type](#) of files in the same way the Unix `file(1)` command works: it looks at the first few bytes of the file. It is intended as a "second line of defense" for cases that `mod_mime` can't resolve.

This module is derived from a free version of the `file(1)` command for Unix, which uses "magic numbers" and other hints from a file's contents to figure out what the contents are. This module is active only if the magic file is specified by the [MimeMagicFile](#) directive.
The contents of the file are plain ASCII text in 4-5 columns. Blank lines are allowed but ignored. Commented lines use a hash mark (#). The remaining lines are parsed for the following columns:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>byte number to begin checking from &quot;&gt;&quot; indicates a dependency upon the previous non-&quot;&gt;&quot; line</td>
</tr>
<tr>
<td>2</td>
<td>type of data to match</td>
</tr>
<tr>
<td></td>
<td>byte</td>
</tr>
<tr>
<td></td>
<td>short</td>
</tr>
<tr>
<td></td>
<td>long</td>
</tr>
<tr>
<td></td>
<td>string</td>
</tr>
<tr>
<td></td>
<td>date</td>
</tr>
<tr>
<td></td>
<td>beshort</td>
</tr>
<tr>
<td></td>
<td>belong</td>
</tr>
<tr>
<td></td>
<td>bedate</td>
</tr>
<tr>
<td></td>
<td>leshort</td>
</tr>
<tr>
<td></td>
<td>lelong</td>
</tr>
<tr>
<td></td>
<td>ledate</td>
</tr>
<tr>
<td>3</td>
<td>contents of data to match</td>
</tr>
<tr>
<td>4</td>
<td>MIME type if matched</td>
</tr>
<tr>
<td>5</td>
<td>MIME encoding if matched (optional)</td>
</tr>
</tbody>
</table>

For example, the following magic file lines would recognize some audio formats:

```
# Sun/NeXT audio data
```
Or these would recognize the difference between *.doc files containing Microsoft Word or FrameMaker documents. (These are incompatible file formats which use the same file suffix.)

An optional MIME encoding can be included as a fifth column. For example, this can recognize gzipped files and set the encoding for them.

# gzip (GNU zip, not to be confused with
# [Info-ZIP/PKWARE] zip archiver)
0 string \037\213 application/octet-stream x-gzip
This module is not for every system. If your system is barely keeping up with its load or if you're performing a web server benchmark, you may not want to enable this because the processing is not free.

However, an effort was made to improve the performance of the original file(1) code to make it fit in a busy web server. It was designed for a server where there are thousands of users who publish their own documents. This is probably very common on intranets. Many times, it's helpful if the server can make more intelligent decisions about a file's contents than the file name allows ...even if just to reduce the "why doesn't my page work" calls when users improperly name their own files. You have to decide if the extra work suits your environment.
The following notes apply to the `mod_mime_magic` module and are included here for compliance with contributors' copyright restrictions that require their acknowledgment.

`mod_mime_magic`: MIME type lookup via file magic numbers
Copyright (c) 1996-1997 Cisco Systems, Inc.

This software was submitted by Cisco Systems to the Apache Group in July 1997. Future revisions and derivatives of this source code must acknowledge Cisco Systems as the original contributor of this module. All other licensing and usage conditions are those of the Apache Group.

Some of this code is derived from the free version of the file command originally posted to comp.sources.unix. Copyright info for that program is included below as required.

- Copyright (c) Ian F. Darwin, 1987. Written by Ian F. Darwin.

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

1. The author is not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.

2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.

3. Altered versions must be plainly marked as such, and must
not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.

4. This notice may not be removed or altered.

For compliance with Mr Darwin's terms: this has been very significantly modified from the free "file" command.

- all-in-one file for compilation convenience when moving from one version of Apache to the next.
- Memory allocation is done through the Apache API's pool structure.
- All functions have had necessary Apache API request or server structures passed to them where necessary to call other Apache API routines. *(i.e., usually for logging, files, or memory allocation in itself or a called function.)*
- struct magic has been converted from an array to a single-ended linked list because it only grows one record at a time, it's only accessed sequentially, and the Apache API has no equivalent of `realloc()`.
- Functions have been changed to get their parameters from the server configuration instead of globals. *(It should be reentrant now but has not been tested in a threaded environment.)*
- Places where it used to print results to stdout now saves them in a list where they're used to set the MIME type in the Apache request record.
- Command-line flags have been removed since they will never be used here.
| **Description:** | Enable MIME-type determination based on file contents using the specified magic file |
| **Syntax:** | MimeMagicFile file-path |
| **Context:** | server config, virtual host |
| **Status:** | Extension |
| **Module:** | mod_mime_magic |

The **MimeMagicFile** directive can be used to enable this module, the default file is distributed at conf/magic. Non-rooted paths are relative to the **ServerRoot**. Virtual hosts will use the same file as the main server unless a more specific setting is used, in which case the more specific setting overrides the main server's file.

**Example**

MimeMagicFile conf/magic
Apache mod_negotiation

Base
negotiation_module
mod_negotiation.c

- (type-map) variants
- MultiViews (MultiViews Option)

Bugfix checklist
httpd changelog
Known issues
Report a bug

Options
mod_mime
RFC 822

**Content-Encoding:**
- Apache AddEncoding compress
- compress gzip x-gzip

**Content-Language:**
(RFC 1766)

**Content-Length:**
()

**Content-Type:**

<table>
<thead>
<tr>
<th>MIME level</th>
<th>text/html 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>qs</td>
<td>0.0 1.0</td>
</tr>
<tr>
<td>variant</td>
<td>ASCII ASCII</td>
</tr>
</tbody>
</table>

Content-Type: image/jpeg; qs=0.8

**URI:**
()

**Body:**
Apache 2.0 Body

**Example:**

Body:----xyz----
<html>
<body>
<p>Content of the page.</p>
</body>
</html>
</body>
</html>
-----xyz-----
MultiViews  Multiviews Options
  /some/dir/foo
CacheNegotiatedDocs: On|Off

Base: mod_negotiation

HTTP/1.0

HTTP/1.1

2.0 CacheNegotiatedDocs on
ForceLanguagePriority

Prefer

Fallback

Prefer

Fallback

Prefer

Fallback

Prefer

Fallback

Prefer

Fallback

Prefer

Fallback
• AddLanguage
LanguagePriority MultiViews

Example:
LanguagePriority en fr de

foo.html  foo.html.fr  foo.html.de
foo.html.fr

- AddLanguage
## Apache Module mod_nw_ssl

<table>
<thead>
<tr>
<th>Description</th>
<th>Enable SSL encryption for NetWare</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Base</td>
</tr>
<tr>
<td>Module Identifier</td>
<td>nwssl_module</td>
</tr>
<tr>
<td>Source File</td>
<td>mod_nw_ssl.c</td>
</tr>
<tr>
<td>Compatibility</td>
<td>NetWare only</td>
</tr>
</tbody>
</table>

### Summary

This module enables SSL encryption for a specified port. It takes advantage of the SSL encryption functionality that is built into the NetWare operating system.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>List of additional client certificates</th>
</tr>
</thead>
</table>
| **Syntax:**     | NWSSLTrustedCerts filename [filename]  
|                 | ... |
| **Context:**    | server config |
| **Status:**     | Base |
| **Module:**     | mod_nw_ssl |

Specifies a list of client certificate files (DER format) that are used when creating a proxied SSL connection. Each client certificate used by a server must be listed separately in its own .der file.
**Description:** Allows a connection to be upgraded to an SSL connection upon request

**Syntax:** NWSSLUpgradeable [IP-address:]portnumber

**Context:** server config

**Status:** Base

**Module:** mod_nw_ssl

Allow a connection that was created on the specified address and/or port to be upgraded to an SSL connection upon request from the client. The address and/or port must have already been defined previously with a `Listen` directive.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Enables SSL encryption for the specified port</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SecureListen [IP-address:]portnumber Certificate-Name [MUTUAL]</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_nw_ssl</td>
</tr>
</tbody>
</table>

Specifies the port and the eDirectory based certificate name that will be used to enable SSL encryption. An optional third parameter also enables mutual authentication.
### Apache Module mod_privileges

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Support for Solaris privileges and for running virtual hosts under different user IDs.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>privileges_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_privileges.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.3 and up, on Solaris 10 and OpenSolaris platforms</td>
</tr>
</tbody>
</table>

### Summary

This module enables different Virtual Hosts to run with different Unix User and Group IDs, and with different Solaris Privileges. In particular, it offers a solution to the problem of privilege separation between different Virtual Hosts, first promised by the abandoned perchild MPM. It also offers other security enhancements.

Unlike perchild, mod_privileges is not itself an MPM. It works within a processing model to set privileges and User/Group per request in a running process. It is therefore not compatible with a threaded MPM, and will refuse to run under one.

mod_privileges raises security issues similar to those of suexec. But unlike suexec, it applies not only to CGI programs but to the entire request processing cycle, including in-process applications and subprocesses. It is ideally suited to running PHP applications under mod_php, which is also incompatible with threaded MPMs. It is also well-suited to other in-process scripting applications such as mod_perl, mod_python, and mod_ruby, and to applications implemented in C as apache modules where privilege separation is an issue.
**mod_privileges** introduces new security concerns in situations where untrusted code may be run **within the webserver process**. This applies to untrusted modules, and scripts running under modules such as mod_php or mod_perl. Scripts running externally (e.g. as CGI or in an appserver behind mod_proxy or mod_jk) are NOT affected.

The basic security concerns with mod_privileges are:

- Running as a system user introduces the same security issues as mod_suexec, and near-equivalents such as cgiwrap and suphp.
- A privileges-aware malicious user extension (module or script) could escalate its privileges to anything available to the httpd process in any virtual host. This introduces new risks if (and only if) mod_privileges is compiled with the `BIG_SECURITY_HOLE` option.
- A privileges-aware malicious user extension (module or script) could escalate privileges to set its user ID to another system user (and/or group).

The **PrivilegesMode** directive allows you to select either **FAST** or **SECURE** mode. You can mix modes, using **FAST** mode for trusted users and fully-audited code paths, while imposing **SECURE** mode where an untrusted user has scope to introduce code.

Before describing the modes, we should also introduce the target use cases: Benign vs Hostile. In a benign situation, you want to separate users for their convenience, and protect them and the server against the risks posed by honest mistakes, but you trust your users are not deliberately subverting system security. In a hostile situation - e.g. commercial hosting - you may have users deliberately attacking the system or each other.
**FAST mode**

In **FAST** mode, requests are run in-process with the selected uid/gid and privileges, so the overhead is negligible. This is suitable for benign situations, but is not secure against an attacker escalating privileges with an in-process module or script.

**SECURE mode**

A request in **SECURE** mode forks a subprocess, which then drops privileges. This is a very similar case to running CGI with suexec, but for the entire request cycle, and with the benefit of fine-grained control of privileges.

You can select different PrivilegesModes for each virtual host, and even in a directory context within a virtual host. **FAST** mode is appropriate where the user(s) are trusted and/or have no privilege to load in-process code. **SECURE** mode is appropriate to cases where untrusted code might be run in-process. However, even in **SECURE** mode, there is no protection against a malicious user who is able to introduce privileges-aware code running before the start of the request-processing cycle.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Determines whether the privileges required by dtrace are enabled.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>DTracePrivileges On</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>DTracePrivileges Off</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_privileges</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available on Solaris 10 and OpenSolaris with non-threaded MPMs (prefork or custom MPM).</td>
</tr>
</tbody>
</table>

This server-wide directive determines whether Apache will run with the privileges required to run dtrace. Note that DTracePrivileges On will not in itself activate DTrace, but DTracePrivileges Off will prevent it working.
| **Description:** | Trade off processing speed and efficiency vs security against malicious privileges-aware code. |
| **Syntax:** | PrivilegesMode FAST | SECURE | SELECTIVE |
| **Default:** | PrivilegesMode FAST |
| **Context:** | server config, virtual host, directory |
| **Status:** | Experimental |
| **Module:** | mod_privileges |
| **Compatibility:** | Available on Solaris 10 and OpenSolaris with non-threaded MPMs (prefork or custom MPM). |

This directive trades off performance vs security against malicious, privileges-aware code. In **SECURE** mode, each request runs in a secure subprocess, incurring a substantial performance penalty. In **FAST** mode, the server is not protected against escalation of privileges as discussed above.

This directive differs slightly between a **<Directory>** context (including equivalents such as Location/Files/If) and a top-level or **<VirtualHost>**.

At top-level, it sets a default that will be inherited by virtualhosts. In a virtual host, **FAST** or **SECURE** mode acts on the entire HTTP request, and any settings in a **<Directory>** context will be **ignored**. A third pseudo-mode **SELECTIVE** defers the choice of **FAST** vs **SECURE** to directives in a **<Directory>** context.

In a **<Directory>** context, it is applicable only where **SELECTIVE** mode was set for the VirtualHost. Only **FAST** or **SECURE** can be set in this context (**SELECTIVE** would be meaningless).

**Warning**
Where SELECTIVE mode is selected for a virtual host, the activation of privileges must be deferred until \textit{after} the mapping phase of request processing has determined what <\texttt{Directory}> context applies to the request. This might give an attacker opportunities to introduce code through a \texttt{RewriteMap} running at top-level or <\texttt{VirtualHost}> context \textit{before} privileges have been dropped and userid/gid set.
**Description:** Determines whether the virtualhost can run subprocesses, and the privileges available to subprocesses.

**Syntax:**
```
VHostCGIMode  On|Off|Secure
```

**Default:**
```
VHostCGIMode  On
```

**Context:** virtual host

**Status:** Experimental

**Module:** mod_privileges

**Compatibility:** Available on Solaris 10 and OpenSolaris with non-threaded MPMs (prefork or custom MPM).

Determines whether the virtual host is allowed to run fork and exec, the privileges required to run subprocesses. If this is set to **Off** the virtualhost is denied the privileges and will not be able to run traditional CGI programs or scripts under the traditional **mod_cgi**, nor similar external programs such as those created by **mod_ext_filter** or **RewriteMap prog**. Note that it does not prevent CGI programs running under alternative process and security models such as **mod_fcgid**, which is a recommended solution in Solaris.

If set to **On** or **Secure**, the virtual host is permitted to run external programs and scripts as above. Setting **VHostCGIMode Secure** has the effect of denying privileges to the subprocesses, as described for **VHostSecure**.
**Description:** Assign arbitrary privileges to subprocesses created by a virtual host.

**Syntax:**

```
VHostPrivs [+-]?privilege-name [[+-]?privilege-name] ...
```

**Default:** None

**Context:** virtual host

**Status:** Experimental

**Module:** mod_privileges

**Compatibility:** Available on Solaris 10 and OpenSolaris with non-threaded MPMs (`prefork` or custom MPM) and when `mod_privileges` is compiled with the `BIG_SECURITY_HOLE` compile-time option.

VHostCGIPrivs can be used to assign arbitrary privileges to subprocesses created by a virtual host, as discussed under VHostCGIMode. Each `privilege-name` is the name of a Solaris privilege, such as `file_setid` or `sys_nfs`.

*privilege-name* may optionally be prefixed by + or -, which will respectively allow or deny a privilege. If used with neither + nor -, all privileges otherwise assigned to the virtualhost will be denied. You can use this to override any of the default sets and construct your own privilege set.

**Security**

This directive can open huge security holes in apache subprocesses, up to and including running them with root-level powers. Do not use it unless you fully understand what you are doing!
**Description:** Sets the Group ID under which a virtual host runs.

**Syntax:** `VHostGroup unix-groupid`

**Default:** Inherits the group id specified in Group

**Context:** virtual host

**Status:** Experimental

**Module:** `mod_privileges`

**Compatibility:** Available on Solaris 10 and OpenSolaris with non-threaded MPMs (prefork or custom MPM).

The `VHostGroup` directive sets the Unix group under which the server will process requests to a virtualhost. The group is set before the request is processed and reset afterwards using Solaris Privileges. Since the setting applies to the process, this is not compatible with threaded MPMs.

`Unix-group` is one of:

- **A group name**
  - Refers to the given group by name.

- **# followed by a group number.**
  - Refers to a group by its number.

**Security**

This directive cannot be used to run apache as root! Nevertheless, it opens potential security issues similar to those discussed in the suexec documentation.

**See also**

- [Group](#)
SuexecUserGroup
**Description:** Assign arbitrary privileges to a virtual host.

**Syntax:**
```plaintext
VHostPrivs [+-]?privilege-name [[+-]?privilege-name] ...
```

**Default:** None

**Context:** virtual host

**Status:** Experimental

**Module:** mod_privileges

**Compatibility:** Available on Solaris 10 and OpenSolaris with non-threaded MPMs (**prefork** or custom MPM) and when `mod_privileges` is compiled with the `BIG_SECURITY_HOLE` compile-time option.

---

**VHostPrivs** can be used to assign arbitrary privileges to a virtual host. Each `privilege-name` is the name of a Solaris privilege, such as `file_setid` or `sys_nfs`.

A `privilege-name` may optionally be prefixed by + or -, which will respectively allow or deny a privilege. If used with neither + nor -, all privileges otherwise assigned to the virtual host will be denied. You can use this to override any of the default sets and construct your own privilege set.

---

**Security**

This directive can open huge security holes in apache, up to and including running requests with root-level powers. Do not use it unless you fully understand what you are doing!
**Description:** Determines whether the server runs with enhanced security for the virtualhost.

**Syntax:** VHostSecure On|Off

**Default:** VHostSecure On

**Context:** virtual host

**Status:** Experimental

**Module:** mod_privileges

**Compatibility:** Available on Solaris 10 and OpenSolaris with non-threaded MPMs (prefork or custom MPM).

Determines whether the virtual host processes requests with security enhanced by removal of Privileges that are rarely needed in a webserver, but which are available by default to a normal Unix user and may therefore be required by modules and applications. It is recommended that you retain the default (On) unless it prevents an application running. Since the setting applies to the process, this is not compatible with threaded MPMs.

**Note**

If VHostSecure prevents an application running, this may be a warning sign that the application should be reviewed for security.
**Description:** Sets the User ID under which a virtual host runs.

**Syntax:** `VHostUser unix-userid`

**Default:** Inherits the userid specified in `User`

**Context:** virtual host

**Status:** Experimental

**Module:** `mod_privileges`

**Compatibility:** Available on Solaris 10 and OpenSolaris with non-threaded MPMs (`prefork` or custom MPM).

The `VHostUser` directive sets the Unix userid under which the server will process requests to a virtualhost. The userid is set before the request is processed and reset afterwards using Solaris Privileges. Since the setting applies to the process, this is not compatible with threaded MPMs.

`Unix-userid` is one of:

- **A username**
  Refers to the given user by name.

- **# followed by a user number.**
  Refers to a user by its number.

**Security**

This directive cannot be used to run apache as root! Nevertheless, it opens potential security issues similar to those discussed in the `suexec` documentation.

**See also**

- [User](#)
- [SuexecUserGroup](#)
Apache mod_proxy

HTTP/1.1
/ Extension
proxy_module
mod_proxy.c

ProxyRequests

Apache / AJP13 (Apache JServe Protocol version 1.3), CONNECT (SSL), HTTP/0.9, HTTP/1.0, HTTP/1.1

Apache mod_proxy : mod_proxy_ftp, mod_proxy_ajp, mod_proxy_balancer, mod_proxy_connect, mod_proxy_connect

mod_c
Bugfix checklist

httpd changelog
Known issues
Report a bug

mod_cache
mod_proxy_http
mod_proxy_ftp
mod_proxy_connect
mod_proxy_balancer
mod_ssl
Apache

ProxyRequests

ProxyPass RewriteRule [P]
mod_cache

ProxyRequests On
ProxyVia On

<Proxy *>
  Order deny,allow
  Deny from all
  Allow from internal.example.com
</Proxy>

ProxyRequests Off

<Proxy *>
  Order deny,allow
  Allow from all
</Proxy>

ProxyPass /foo http://foo.example.com/bar
ProxyPassReverse /foo http://foo.example.com/bar
<Proxy>

<Proxy */>
    Order Deny,Allow
    Deny from all
    Allow from 192.168.0
</Proxy>

mod_authz_host

(ProxyRequests )
(ProxyRequests Off  ProxyPass )
ProxyBlock
Apache

WWW
"http://somehost/"

Apache
Keepalive HTTP/1.1
keepalive

table force-proxy-request-1.0 proxy-nokeepalive

<Location /buggyappserver/>
  ProxyPass http://buggyappserver:7001/foo/
  SetEnv force-proxy-request-1.0 1
  SetEnv proxy-nokeepalive 1
</Location>
POST

mod_proxy_http

Content-Length
chunked

Length

HTTP chunked

proxy-sendcl
<table>
<thead>
<tr>
<th></th>
<th>Number of additional Balancers that can be added Post-configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BalancerGrowth #</td>
</tr>
<tr>
<td></td>
<td>BalancerGrowth 5</td>
</tr>
<tr>
<td></td>
<td>Extension</td>
</tr>
<tr>
<td></td>
<td>mod_proxy</td>
</tr>
<tr>
<td></td>
<td>BalancerGrowth is only available in Apache HTTP Server</td>
</tr>
<tr>
<td></td>
<td>2.3.13 and later.</td>
</tr>
</tbody>
</table>
BalancerInherit: Inherit ProxyPassed Balancers/Workers from the main server
BalancerInherit On|Off
BalancerInherit On

Extension
mod_proxy
BalancerInherit is only available in Apache HTTP Server 2.4.5 and later.
Add a member to a load balancing group

Extension

mod_proxy

Documentation not yet translated. Please see English version of document.
BalancerPersist

Attempt to persist changes made by the Balancer Manager across restarts.

BalancerPersist On|Off

BalancerPersist Off

Extension

mod_proxy

BalancerPersist is only available in Apache HTTP Server 2.4.4 and later.
NoProxy

Apache

ProxyRemote

NoProxy  * http://firewall.mycompany.com:81
NoProxy  .mycompany.com 192.168.112.0/21

NoProxy  host  :

Domain

Domain  DNS

Domain  Hostname (DNS !)  Domain

SubNet

SubNet ()
(8)
192.168 192.168.0.0
    192.168.0.0 16 (255.255.0.0)

192.168.112.0/21
    192.168.112.0/21 21 (255.255.248.0 32 SubNet IPAddr 0 SubNet (0.0.0.0/0 IP _Default_

**IPAddr**

IPAddr ()

| 192.168.123.7 |

**Hostname**

Hostname DNS IPAddr apache

| prep.ai.mit.edu www.apache.org |

**Hostname**

Hostname IPAddr DNS Apache

| WWW.MyDomain.com www.mydomain.com. () |
• DNS
<Proxy>

YOURNETWORK.EXAMPLE.COM :

<Proxy *>
  Order Deny, Allow
  Deny from all
  Allow from yournetwork.example.com
</Proxy>

EXAMPLE.COM  FOO

<Proxy http://example.com/foo/*>
  SetOutputFilter INCLUDES
</Proxy>
Add proxy information in X-Forwarded-* headers
ProxyAddHeaders Off|On
ProxyAddHeaders On

Extension
mod_proxy
Available in version 2.3.10 and later
ProxyBadHeader | IsError | Ignore | StartBody
--- | --- | --- | ---
 ProxyBadHeader | IsError

Extension: mod_proxy
Version: 2.0.44

ProxyBadHeader ( )

**IsError**
- 502 (Bad Gateway)

**Ignore**

**StartBody**
ProxyBlock

Example
ProxyBlock joes-garage.com some-host.co.uk rocky.wotsamattau.edu

rocky.wotsamattau.edu IP
wotsamattau.edu wotsamattau

ProxyBlock *

Apache

ProxyRemote * http://firewall.mycompany.com:81
NoProxy .mycompany.com 192.168.112.0/21
ProxyDomain .mycompany.com
ProxyErrorOverride On|Off
ProxyErrorOverride Off

Extension
mod_proxy
2.0
ProxyIOBufferSize

ProxyIOBufferSize bytes

ProxyIOBufferSize 8192

, 

Extension

mod_proxy
<ProxyMatch regex> ...
</ProxyMatch>

Extension

mod_proxy

<ProxyMatch> URL

<Proxy>
ProxyMaxForwards number
ProxyMaxForwards 10

Extension
mod_proxy
Apache 2.0

ProxyMaxForwards Max-Forwards
ProxyMaxForwards 15
ProxyPass

http://example.com/

ProxyPass /mirror/foo/ http://backend.example.com/

http://example.com/mirror/foo/bar
http://backend.example.com/bar

ProxyPass /mirror/foo/i !
ProxyPass /mirror/foo http://backend.example.com

/mirror/foo/i /mirror/foo

2.1
prefork MPM 1 worker MPM

ThreadsPerChild
ProxyPass /example http://backend.example.com smax=5 max=20
ttl=120 retry=300

min 0
max 1...n Hard Maximum
(smax max)

smax max Soft Maximum (: )
ttl

retry 60
loadfactor 1 BalancerMember
route -
redirect -

Proxy balancer://

lbmethod - Balancer
bytraffic
byrequest
sticky session -
JSESSIONID PHPSESSIONID

nofailover Off On
timeout 0
maxattempts 1

ProxyPass /special-area http://special.example.com/ smax=5
max=10
ProxyPass / balancer://mycluster stickysession=jsessionid
nofailover=On

<Proxy balancer://mycluster>
  BalancerMember http://1.2.3.4:8009
  BalancerMember http://1.2.3.5:8009 smax=10
  # Less powerful server, don't send as many requests there
  BalancerMember http://1.2.3.6:8009 smax=1 loadfactor=20
</Proxy>

<Location>

  [P] RewriteRule

</Location>
<table>
<thead>
<tr>
<th>ProxyPassInherit</th>
<th>Inherit ProxyPass directives defined from the main server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Off</td>
<td>ProxyPassInherit On</td>
</tr>
<tr>
<td></td>
<td>ProxyPassInherit On</td>
</tr>
</tbody>
</table>

Extension `mod_proxy` supported.

ProxyPassInherit is only available in Apache HTTP Server 2.4.5 and later.
Enable Environment Variable interpolation in Reverse Proxy configurations.

Extension: mod_proxy

Documentation not yet translated. Please see English version of document.
Maps remote servers into the local server URL-space using regular expressions.

Extension

mod_proxy

Documentation not yet translated. Please see English version of document.
ProxyPassReverse [path] url

Extension

Apache HTTP

Location, Content-Location, mod_proxy

Apache

HTTP

Nick Kew

mod_proxy_html

path  url  URL

http://example.com/

ProxyPass /mirror/foo/ http://backend.example.com/
ProxyPassReverse /mirror/foo/ http://backend.example.com/
ProxyPassReverseCookieDomain backend.example.com
public.example.com
ProxyPassReverseCookiePath / /mirror/foo/

http://example.com/mirror/foo/bar
http://backend.example.com/bar
) backend.example.com
http://backend.example.com/bar
http://backend.example.com/quux
http://example.com/mirror/foo/quux

UseCanonicalName

ProxyPassReverse ProxyPass
(RewriteRule ... [P])

<Location>
Set-Cookie Domain
ProxyPassReverseCookieDomain internal-domain
  public-domain

Extension
mod_proxy

ProxyPassReverse URL Set-Cookie domain
<table>
<thead>
<tr>
<th>Reverse Set-Cookie Path</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ProxyPassReverseCookiePath</strong> <em>internal-path</em></td>
</tr>
<tr>
<td><em>public-path</em></td>
</tr>
<tr>
<td>, ,</td>
</tr>
<tr>
<td><strong>Extension</strong></td>
</tr>
<tr>
<td><strong>mod_proxy</strong></td>
</tr>
</tbody>
</table>

**ProxyPassReverse** URL Set-Cookie path
ProxyPreserveHost: Off

ProxyPass Host: Off
<table>
<thead>
<tr>
<th>Protocol</th>
<th>ProxyReceiveBufferSize</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP</td>
<td>ProxyReceiveBufferSize bytes</td>
<td>0</td>
</tr>
<tr>
<td>FTP</td>
<td>ProxyReceiveBufferSize</td>
<td>2048</td>
</tr>
</tbody>
</table>

**Extension**

- `mod_proxy`
ProxyRemote match remote-server

Extension
mod_proxy

match URL

remote-server = scheme://hostname[:port]

scheme

ProxyRemote * http://cleversite.com

FTP HTTP
ProxyRemoteMatch regex remote-server

Extension
mod_proxy

ProxyRemoteMatch URL Pr
ProxyRequests On|Off

ProxyRequests Off

Extension

mod_proxy

Apache

Off

HTTP FTP

(ProxyRequests

mod_proxy.h

ProxyRequests
Set various Proxy balancer or member parameters

Extension

mod_proxy

Documentation not yet translated. Please see English version of document.
Set local IP address for outgoing proxy connections

ProxySourceAddress *address*

Extension

mod_proxy

Available in version 2.3.9 and later
Show Proxy LoadBalancer status in mod_status

Extension
mod_proxy

Documentation not yet translated. Please see English version of document.
ProxyTimeout seconds
ProxyTimeout 300

Extension
mod_proxy
Apache 2.0.31
ProxyVia On|Off|Full|Block
ProxyVia Off

Extension
mod_proxy

Via: HTTP
14.45

- Off
- On
- Full
- Block
## Apache mod_proxy_ajp

<table>
<thead>
<tr>
<th>mod_proxy</th>
<th>AJP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extension</td>
<td></td>
</tr>
<tr>
<td>proxy_ajp_module</td>
<td></td>
</tr>
<tr>
<td>mod_proxy_ajp.c</td>
<td></td>
</tr>
</tbody>
</table>

**mod_proxy** Apache JServ Protocol version 1.3 (AJP13)

### Bugfix checklist

- [httpd changelog](httpd_changelog)
- [Known issues](Known_issues)
- [Report a bug](Report_a_bug)

**mod_proxy**
AJP13

- Idle
- Assigned

(HTTP > 0)

- SEND_HEADERS
- SEND_BODY_CHUNK
- GET_BODY_CHUNK
- END_RESPONSE
XDR  

: 

4 : byte, boolean, integer, string

**Byte**

**Boolean**

\[ 1 = true, 0 = false \] (C)

**Integer**

\[ 0 \quad 2^{16} (32768) \quad 2 \]

**String**

\[ (2^{16}) 2 \] (\'\n
\[ strlen \]

\[ \backslash 0 \ C \]

\[ 8 \times 1024 \]

\[ 0x1234 \quad AB \text{ (ASCII ')} \]

\[ ) \quad 2^{16} \]

**Server->Container**

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4...(n+3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>0x12</td>
<td>0x34</td>
<td>(n)</td>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>

**Container->Server**

<table>
<thead>
<tr>
<th>Byte</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4...(n+3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contents</td>
<td>A</td>
<td>B</td>
<td>(n)</td>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
Data   Forward Request

3  Send Body Chunk
4  Send Headers  
5  End Response  
6  Get Body Chunk
9  CPong   CPong

none Data  (2 )
Forward Request

```
AJP13_FORWARD_REQUEST :=
  prefix_code  (byte) 0x02 = JK_AJP13_FORWARD_REQUEST
  method      (byte)
  protocol    (string)
  req_uri     (string)
  remote_addr (string)
  remote_host (string)
  server_name (string)
  server_port (integer)
  is_ssl      (boolean)
  num_headers (integer)
  request_headers *(req_header_name req_header_value)
  attributes    *(attribut_name attribute_value)
  request_terminator (byte) 0xFF
```

```
request_headers :

  req_header_name :=
    sc_req_header_name | (string) [see below for how this is parsed]

  sc_req_header_name := 0xA0xx (integer)

  req_header_value := (string)

:

attribute_name := sc_a_name | (sc_a_req_attribute string)

attribute_value := (string)

content-length
```

Forward Request
Request prefix

2  Prefix
## Method

HTTP 1:

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>OPTIONS</td>
<td>1</td>
</tr>
<tr>
<td>GET</td>
<td>2</td>
</tr>
<tr>
<td>HEAD</td>
<td>3</td>
</tr>
<tr>
<td>POST</td>
<td>4</td>
</tr>
<tr>
<td>PUT</td>
<td>5</td>
</tr>
<tr>
<td>DELETE</td>
<td>6</td>
</tr>
<tr>
<td>TRACE</td>
<td>7</td>
</tr>
<tr>
<td>PROPFIND</td>
<td>8</td>
</tr>
<tr>
<td>PROPPATCH</td>
<td>9</td>
</tr>
<tr>
<td>MKCOL</td>
<td>10</td>
</tr>
<tr>
<td>COPY</td>
<td>11</td>
</tr>
<tr>
<td>MOVE</td>
<td>12</td>
</tr>
<tr>
<td>LOCK</td>
<td>13</td>
</tr>
<tr>
<td>UNLOCK</td>
<td>14</td>
</tr>
<tr>
<td>ACL</td>
<td>15</td>
</tr>
<tr>
<td>REPORT</td>
<td>16</td>
</tr>
<tr>
<td>VERSION-CONTROL</td>
<td>17</td>
</tr>
<tr>
<td>CHECKIN</td>
<td>18</td>
</tr>
<tr>
<td>CHECKOUT</td>
<td>19</td>
</tr>
<tr>
<td>UNCHECKOUT</td>
<td>20</td>
</tr>
<tr>
<td>SEARCH</td>
<td>21</td>
</tr>
<tr>
<td>MKWORKSPACE</td>
<td>22</td>
</tr>
<tr>
<td>UPDATE</td>
<td>23</td>
</tr>
<tr>
<td>LABEL</td>
<td>24</td>
</tr>
<tr>
<td>MERGE</td>
<td>25</td>
</tr>
<tr>
<td>BASELINE_CONTROL</td>
<td>26</td>
</tr>
</tbody>
</table>
protocol, req_uri, remote_addr, remote_host, server_name, server_port, is_ssl

Headers
request_headers:

num_header
req_header_name/ req_header_value

accept 0xA001 SC_REQ_ACCEPT
accept-charset 0xA002 SC_REQ_ACCEPT_CHARSET
accept-encoding 0xA003 SC_REQ_ACCEPT_ENCODING
accept-language 0xA004 SC_REQ_ACCEPT_LANGUAGE
authorization 0xA005 SC_REQ_AUTHORIZATION
connection 0xA006 SC_REQ_CONNECTION
content-type 0xA007 SC_REQ_CONTENT_TYPE
content-length 0xA008 SC_REQ_CONTENT_LENGTH
cookie 0xA009 SC_REQ_COOKIE
cookie2 0xA00A SC_REQ_COOKIE2
host 0xA00B SC_REQ_HOST
pragma 0xA00C SC_REQ_PRAGMA
referer 0xA00D SC_REQREFERER
user-agent 0xA00E SC_REQ_USER_AGENT

Java 2
<table>
<thead>
<tr>
<th>Information</th>
<th>Code Value</th>
<th>Type Of Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>?context</td>
<td>0x01</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>0x02</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>servlet_path</td>
<td>0x03</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>0x03</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>remote_user</td>
<td>0x04</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>0x05</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>query_string</td>
<td>0x06</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>0x07</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>ssl_cert</td>
<td>0x08</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>?</td>
<td>0x09</td>
<td>String</td>
<td></td>
</tr>
<tr>
<td>ssl_cipher</td>
<td>0x0A</td>
<td>String</td>
<td>Name (the name of the attribute follows)</td>
</tr>
<tr>
<td>?</td>
<td>0x0B</td>
<td>Integer</td>
<td></td>
</tr>
<tr>
<td>ssl_key_size</td>
<td>0xFF</td>
<td>-</td>
<td>request_terminator</td>
</tr>
</tbody>
</table>

context servlet_path C Java
remote_user  auth_type      HTTP

query_string, ssl_cert, ssl_cipher, ssl_session
HTTP HTTPS

jvm_route ——— Tomcat

req_attribute  0x0A

0xFF
AJP13_SEND_BODY_CHUNK :=
  prefix_code  3
  chunk_length  (integer)
  chunk  *(byte)
  chunk_terminator  (byte) 0x00

AJP13_SEND_HEADERS :=
  prefix_code  4
  http_status_code  (integer)
  http_status_msg  (string)
  num_headers  (integer)
  response_headers *(res_header_name header_value)

  res_header_name :=
    sc_res_header_name  [string]  [see below for how this is parsed]

  sc_res_header_name := 0xA0  (byte)

  header_value :=  (string)

AJP13_END_RESPONSE :=
  prefix_code  5
  reuse  (boolean)

AJP13_GET_BODY_CHUNK :=
  prefix_code  6
  requested_length  (integer)

---

Send Body Chunk

Send Headers

HTTP (200 OK):
End Response
  reuse  (==1) TCP

Get Body Chunk

  (               )
  - 6 ))
  (               )
Apache mod_proxy_balancer

mod_proxy HTTP, FTP AJP13

mod_proxy mod_proxy_balancer

Bugfix checklist

httpd changelog
Known issues
Report a bug

mod_proxy
2
Counting) 1bmethod
lbmethod=byrequests

lbfactor

lbstatus

for each worker in workers
    worker lbstatus += worker lbfactor
    total factor += worker lbfactor
    if worker lbstatus > candidate lbstatus
        candidate = worker

candidate lbstatus -= total factor

: 

<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbfactor</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>lbstatus</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

b

<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbstatus</td>
<td>-50</td>
<td>0</td>
<td>25</td>
<td>25</td>
</tr>
<tr>
<td>lbstatus</td>
<td>-25</td>
<td>0</td>
<td>-25</td>
<td>50</td>
</tr>
<tr>
<td>lbstatus</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

(repeat)

: a c d a c d a c d ... :
<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbfactor</td>
<td>25</td>
<td>25</td>
<td>25</td>
<td>25</td>
</tr>
</tbody>
</table>

: 

<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbfactor</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

This is because all values of *lbfactor* are normalized with respect to the others. For:

```
lbfactor : 
```

```
<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbfactor</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

```
b a c 4
```

: 

```
<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbfactor</td>
<td>70</td>
</tr>
</tbody>
</table>
```  

```
lbstatus | -30 | 30 |
|          | 40  | -40|
|          | 10  | -10|
|          | -20 | 20 |
|          | -50 | 50 |
|          | 20  | -20|
|          | -10 | 10 |
|          | -40 | 40 |
|          | 30  | -30|
|          | 0   | 0  |
| 10 | a | 7 | b | 3 |

(repeat)
Weighted Traffic Counting

lbmethod=bytraffic  Request Counting

lbfactor

<table>
<thead>
<tr>
<th>worker</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>lbfactor</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

| b | a | c | 2 |

| b | 2 | I/O |
<Location /balancer-manager>
SetHandler balancer-manager

Order Deny,Allow
Deny from all
Allow from .foo.com
</Location>

http://your.server.name/balancer-manager
Apache mod_proxy_connect

CONNECT
mod_proxy
Extension
proxy_connect_module
mod_proxy_connect.c

mod_proxy CONNECT HTTP
SSL

CONNECT
mod_proxy mod_proxy_connect

Bugfix checklist
httpd changelog
Known issues
Report a bug
AllowCONNECT
mod_proxy
AllowCONNECT ports that are allowed to CONNECT through the proxy:

```
AllowCONNECT port[-port] [port[-port]] ...
```

```
AllowCONNECT 443 563
```

**Extension**

**mod_proxy_connect**

Moved from [mod_proxy](https://httpd.apache.org/docs/2.4/mod/mod_proxy.html) in Apache 2.3.5. Port ranges available since Apache 2.3.7.
Apache Module mod_proxy_express

**Description:** Dynamic mass reverse proxy extension for mod_proxy

**Status:** Extension

**Module Identifier:** proxy_express_module

**Source File:** mod_proxy_express.c

**Summary**

This module creates dynamically configured mass reverse proxies, by mapping the Host: header of the HTTP request to a server name and backend URL stored in a DBM file. This allows for easy use of a huge number of reverse proxies with no configuration changes. It is much less feature-full than mod_proxy_balancer, which also provides dynamic growth, but is intended to handle much, much larger numbers of backends. It is ideally suited as a front-end HTTP switch and for micro-services architectures.

This module *requires* the service of mod_proxy.

**Warning**

Do not enable proxying until you have secured your server. Open proxy servers are dangerous both to your network and to the Internet at large.

**Limitations**

- This module is not intended to replace the dynamic capability of mod_proxy_balancer. Instead, it is intended to be mostly a lightweight and fast alternative to using mod_rewrite with RewriteMap and the [P] flag for mapped reverse proxying.
- It does not support regex or pattern matching at all.
It emulates:

```xml
<VirtualHost *:80>
    ServerName front.end.server
    ProxyPass "/" "back.end.server:port"
    ProxyPassReverse "/" "back.end.server:port"
</VirtualHost>
```

That is, the entire URL is appended to the mapped backend URL. This is in keeping with the intent of being a simple but fast reverse proxy switch.

Bugfix checklist

- [httpd changelog](#)
- [Known issues](#)
- [Report a bug](#)

See also

- [mod_proxy](#)
- [BalancerMember](#)
- [BalancerGrowth](#)
- [BalancerPersist](#)
- [BalancerInherit](#)
The **ProxyExpressDBMFile** directive points to the location of the Express map DBM file. This file serves to map the incoming server name, obtained from the Host: header, to a backend URL.

### Note

The file is constructed from a plain text file format using the **httxt2dbm** utility.

#### ProxyExpress map file

```plaintext
##
##express-map.txt:
##

www1.example.com http://192.168.211.2:8080
www2.example.com http://192.168.211.12:8088
www3.example.com http://192.168.212.10
```

#### Create DBM file

```
httxt2dbm -i express-map.txt -o emap
```

#### Configuration

```
ProxyExpressEnable on
ProxyExpressDBMFile emap
```
The **ProxyExpressDBMType** directive controls the DBM type expected by the module. The default is the default DBM type created with `httxt2dbm`.

Possible values are (not all may be available at run time):

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>db</td>
<td>Berkeley DB files</td>
</tr>
<tr>
<td>gdbm</td>
<td>GDBM files</td>
</tr>
<tr>
<td>ndbm</td>
<td>NDBM files</td>
</tr>
<tr>
<td>sdbm</td>
<td>SDBM files (always available)</td>
</tr>
<tr>
<td>default</td>
<td>default DBM type</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
<td>Enable the module functionality.</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td><strong>Syntax:</strong></td>
<td>ProxyExpressEnable [on</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>off</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_proxy_express</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.3.13 and later</td>
</tr>
</tbody>
</table>

The `ProxyExpressEnable` directive controls whether the module will be active.
Apache Module mod_proxy_fcgi

<table>
<thead>
<tr>
<th>Description:</th>
<th>FastCGI support module for mod_proxy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Extension</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>proxy_fcgi_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_proxy_fcgi.c</td>
</tr>
<tr>
<td>Compatibility:</td>
<td>Available in version 2.3 and later</td>
</tr>
</tbody>
</table>

Summary
This module *requires* the service of mod_proxy. It provides support for the FastCGI protocol.

Thus, in order to get the ability of handling the FastCGI protocol, mod_proxy and mod_proxy_fcgi have to be present in the server.

Unlike mod_fcgid and mod_fastcgi, mod_proxy_fcgi has no provision for starting the application process; fcgistarter is provided (on some platforms) for that purpose. Alternatively, external launching or process management may be available in the FastCGI application framework in use.

Warning
Do not enable proxying until you have secured your server. Open proxy servers are dangerous both to your network and to the Internet at large.
Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

cfgistarter
mod_proxy
mod_authnz_fcgi
Remember, in order to make the following examples work, you have to enable `mod_proxy` and `mod_proxy_fcgi`.

**Single application instance**

```bash
ProxyPass "/myapp/" "fcgi://localhost:4000/"
```

`mod_proxy_fcgi` disables connection reuse by default, so after a request has been completed the connection will NOT be held open by that httpd child process and won't be reused. If the FastCGI application is able to handle concurrent connections from httpd, you can opt-in to connection reuse as shown in the following example:

**Single application instance, connection reuse (2.4.11 and later)**

```bash
ProxyPass "/myapp/" "fcgi://localhost:4000/" enablereuse=on
```

The following example passes the request URI as a filesystem path for the PHP-FPM daemon to run. The request URL is implicitly added to the 2nd parameter. The hostname and port following `fcgi://` are where PHP-FPM is listening. Connection pooling/reuse is enabled.

**PHP-FPM**

```bash
ProxyPassMatch "^/myapp/.*\.php(/.*)?$" "fcgi://localhost:9000/"
```

The following example passes the request URI as a filesystem path for the PHP-FPM daemon to run. In this case, PHP-FPM is listening on a unix domain socket (UDS). Requires 2.4.9 or later. With this syntax, the hostname and optional port following `fcgi://` are ignored.
The balanced gateway needs `mod_proxy_balancer` and at least one load balancer algorithm module, such as `mod_lbmethod_byrequests`, in addition to the proxy modules listed above. `mod_lbmethod_byrequests` is the default, and will be used for this example configuration.

Balanced gateway to multiple application instances

```apache
ProxyPass "/*/myapp/" "balancer://myappcluster/"
<Proxy "balancer://myappcluster/">
    BalancerMember "fcgi://localhost:4000"
    BalancerMember "fcgi://localhost:4001"
</Proxy>
```

You can also force a request to be handled as a reverse-proxy request, by creating a suitable Handler pass-through. The example configuration below will pass all requests for PHP scripts to the specified FastCGI server using reverse proxy. This feature is available in Apache HTTP Server 2.4.10 and later. For performance reasons, you will want to define a `worker` representing the same `fcgi://` backend. The benefit of this form is that it allows the normal mapping of URI to filename to occur in the server, and the local filesystem result is passed to the backend. When FastCGI is configured this way, the server can calculate the most accurate PATH_INFO.

Proxy via Handler

```apache
<FilesMatch "\.php$">
    # Note: The only part that varies is /path/to/app.sock
    SetHandler "proxy:unix:/path/to/app.sock|fcgi://localhost/"
</FilesMatch>

# Define a matching worker.
# The part that is matched to the SetHandler is the part that
# follows the pipe. If you need to distinguish, "localhost; can
# be anything unique.
<Proxy "fcgi://localhost/" enableuse=on max=10>
</Proxy>

<FilesMatch ...
    SetHandler "proxy:fcgi://localhost:9000"
</FilesMatch>

<FilesMatch ...
    SetHandler "proxy:balancer://myappcluster/"
</FilesMatch>
In addition to the configuration directives that control the behaviour of `mod_proxy`, there are a number of *environment variables* that control the FCGI protocol provider:

**proxy-fcgi-pathinfo**

When configured via `ProxyPass` or `ProxyPassMatch`, `mod_proxy_fcgi` will not set the `PATH_INFO` environment variable. This allows the backend FCGI server to correctly determine `SCRIPT_NAME` and `Script-URI` and be compliant with RFC 3875 section 3.3. If instead you need `mod_proxy_fcgi` to generate a "best guess" for `PATH_INFO`, set this env-var. This is a workaround for a bug in some FCGI implementations. This variable can be set to multiple values to tweak at how the best guess is chosen (in 2.4.11 and later only):

**first-dot**

- `PATH_INFO` is split from the slash following the *first* "." in the URL.

**last-dot**

- `PATH_INFO` is split from the slash following the *last* "." in the URL.

**full**

- `PATH_INFO` is calculated by an attempt to map the URL to the local filesystem.

**unescape**

- `PATH_INFO` is the path component of the URL, unescaped / decoded.

**any other value**

- `PATH_INFO` is the same as the path component of the URL. Originally, this was the only proxy-fcgi-pathinfo option.
**Description:** Specify the type of backend FastCGI application

**Syntax:**
```
ProxyFCGIBackendType  FPM|GENERIC
```

**Default:**
```
ProxyFCGIBackendType  FPM
```

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_proxy_fcgi

**Compatibility:** Available in version 2.4.26 and later

This directive allows the type of backend FastCGI application to be specified. Some FastCGI servers, such as PHP-FPM, use historical quirks of environment variables to identify the type of proxy server being used. Set this directive to "GENERIC" if your non PHP-FPM application has trouble interpreting environment variables such as SCRIPT_FILENAME or PATH_TRANSLATED as set by the server.

One example of values that change based on the setting of this directive is SCRIPT_FILENAME. When using `mod_proxy_fcgi` historically, SCRIPT_FILENAME was prefixed with the string "proxy:fcgi://". This variable is what some generic FastCGI applications would read as their script input, but PHP-FPM would strip the prefix then remember it was talking to Apache. In 2.4.21 through 2.4.25, this prefix was automatically stripped by the server, breaking the ability of PHP-FPM to detect and interoperate with Apache in some scenarios.
**Description:** Allow variables sent to FastCGI servers to be fixed up

**Syntax:**
```
ProxyFCGISetEnvIf conditional-expression [!]environment-variable-name [value-expression]
```

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_proxy_fcgi

**Compatibility:** Available in version 2.4.26 and later

Just before passing a request to the configured FastCGI server, the core of the web server sets a number of environment variables based on details of the current request. FastCGI programs often uses these environment variables as inputs that determine what underlying scripts they will process, or what output they directly produce.

Examples of noteworthy environment variables are:

- SCRIPT_NAME
- SCRIPT_FILENAME
- REQUEST_URI
- PATH_INFO
- PATH_TRANSLATED

This directive allows the environment variables above, or any others of interest, to be overridden. This directive is evaluated after the initial values for these variables are set, so they can be used as input into both the condition expressions and value expressions.

Parameter syntax:

```
conditional-expression
```
Specifies an expression that controls whether the environment variable that follows will be modified. For information on the expression syntax, see the examples that follow or the full specification at the `ap_expr` documentation.

**environment-variable-name**

Specifies the CGI environment variable to change, such as PATH_INFO. If preceded by an exclamation point, the variable will be unset.

**value-expression**

Specifies the replacement value for the preceding environment variable. Backreferences, such as "$1", can be included from regular expression captures in *conditional-expression*. If omitted, the variable is set (or overridden) to an empty string — but see the Note below.

```bash
# A basic, unconditional override
ProxyFCGISetEnvIf "true" PATH_INFO "/example"

# Use an environment variable in the value
ProxyFCGISetEnvIf "true" PATH_INFO "%{reqenv:SCRIPT_NAME}"  

# Use captures in the conditions and backreferences in the repl
ProxyFCGISetEnvIf "reqenv('PATH_TRANSLATED') =~ m|/.*prefix|/$1/$3"
```

**Note: Unset vs. Empty**

The following will unset VARIABLE, preventing it from being sent to the FastCGI server:

```bash
ProxyFCGISetEnvIf true !VARIABLE
```

Whereas the following will erase any existing value of VARIABLE (by setting it to the empty string), but the empty VARIABLE will still be sent to the server:

```bash
ProxyFCGISetEnvIf true VARIABLE
```
The CGI/1.1 specification does not distinguish between a variable with an empty value and a variable that does not exist. However, many CGI and FastCGI implementations distinguish (or allow scripts to distinguish) between the two. The choice of which to use is dependent upon your implementation and your reason for modifying the variable.
Apache Module mod_proxy_fdpass

**Description:** fdpass external process support module for mod_proxy

**Status:** Extension

**Module Identifier:** proxy_fdpass_module

**Source File:** mod_proxy_fdpass.c

**Compatibility:** Available for unix in version 2.3 and later

**Summary**

This module *requires* the service of mod_proxy. It provides support for the passing the socket of the client to another process.

mod_proxy_fdpass uses the ability of AF_UNIX domain sockets to pass an open file descriptor to allow another process to finish handling a request.

The module has a proxy_fdpass_flusher provider interface, which allows another module to optionally send the response headers, or even the start of the response body. The default flush provider disables keep-alive, and sends the response headers, letting the external process just send a response body.

In order to use another provider, you have to set the flusher parameter in the ProxyPass directive.

At this time the only data passed to the external process is the client socket. To receive a client socket, call recvfrom with an allocated struct cmsghdr. Future versions of this module may include more data after the client socket, but this is not implemented at this time.
Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod_proxy
Apache Module mod_proxy_ftp

**Description:** FTP support module for mod_proxy

**Status:** Extension

**Module Identifier:** proxy_ftp_module

**Source File:** mod_proxy_ftp.c

**Summary**

This module *requires* the service of mod_proxy. It provides support for the proxying FTP sites. Note that FTP support is currently limited to the GET method.

Thus, in order to get the ability of handling FTP proxy requests, mod_proxy and mod_proxy_ftp have to be present in the server.

**Warning**

Do not enable proxying until you have secured your server. Open proxy servers are dangerous both to your network and to the Internet at large.

**Bugfix checklist**

- [httpd changelog](#)
- [Known issues](#)
- [Report a bug](#)

**See also**
mod_proxy
Why doesn't file type xxx download via FTP?

You probably don't have that particular file type defined as application/octet-stream in your proxy's mime.types configuration file. A useful line can be

```
application/octet-stream  bin dms lha lzh exe class tgz taz
```

Alternatively you may prefer to default everything to binary:

```
ForceType application/octet-stream
```
In the rare situation where you must download a specific file using the FTP ASCII transfer method (while the default transfer is in binary mode), you can override mod_proxy's default by suffixing the request with ;type=a to force an ASCII transfer. (FTP Directory listings are always executed in ASCII mode, however.)
Currently, only GET is supported for FTP in mod_proxy. You can of course use HTTP upload (POST or PUT) through an Apache proxy.
How can I access FTP files outside of my home directory?

An FTP URI is interpreted relative to the home directory of the user who is logging in. Alas, to reach higher directory levels you cannot use ../, as the dots are interpreted by the browser and not actually sent to the FTP server. To address this problem, the so-called Squid %2f hack was implemented in the Apache FTP proxy; it is a solution which is also used by other popular proxy servers like the Squid Proxy Cache. By prepending /%2f to the path of your request, you can make such a proxy change the FTP starting directory to / (instead of the home directory). For example, to retrieve the file /etc/motd, you would use the URL:

ftp://user@host/%2f/etc/motd
How can I hide the FTP cleartext password in my browser's URL line?

To log in to an FTP server by username and password, Apache uses different strategies. In absence of a user name and password in the URL altogether, Apache sends an anonymous login to the FTP server, i.e.,

```
user: anonymous
password: apache_proxy@
```

This works for all popular FTP servers which are configured for anonymous access.

For a personal login with a specific username, you can embed the user name into the URL, like in:

```
ftp://username@host/myfile
```

If the FTP server asks for a password when given this username (which it should), then Apache will reply with a 401 (Authorization required) response, which causes the Browser to pop up the username/password dialog. Upon entering the password, the connection attempt is retried, and if successful, the requested resource is presented. The advantage of this procedure is that your browser does not display the password in cleartext (which it would if you had used

```
ftp://username:password@host/myfile
```

in the first place).

**Note**

The password which is transmitted in such a way is not encrypted on its way. It travels between your browser and the
Apache proxy server in a base64-encoded cleartext string, and between the Apache proxy and the FTP server as plaintext. You should therefore think twice before accessing your FTP server via HTTP (or before accessing your personal files via FTP at all!) When using insecure channels, an eavesdropper might intercept your password on its way.
Why do I get a file listing when I expected a file to be downloaded?

In order to allow both browsing the directories on an FTP server and downloading files, Apache looks at the request URL. If it looks like a directory, or contains wildcard characters ("*?[~"), then it guesses that a listing is wanted instead of a download.

You can disable the special handling of names with wildcard characters. See the `ProxyFtpListOnWildcard` directive.
**Description:** Define the character set for proxied FTP listings

**Syntax:**
```
ProxyFtpDirCharset character set
```

**Default:**
```
ProxyFtpDirCharset ISO-8859-1
```

**Context:**
```
server config, virtual host, directory
```

**Status:**
```
Extension
```

**Module:**
```
mod_proxy_ftp
```

**Compatibility:** Available in Apache 2.2.7 and later. Moved from `mod_proxy` in Apache 2.3.5.

The `ProxyFtpDirCharset` directive defines the character set to be set for FTP directory listings in HTML generated by `mod_proxy_ftp`. 
**Description:** Whether wildcards in requested filenames are escaped when sent to the FTP server

**Syntax:** ProxyFtpEscapeWildcards [on|off]

**Default:** on

**Context:** server config, virtual host, directory

**Status:** Extension

**Module:** mod_proxy_ftp

**Compatibility:** Available in Apache 2.3.3 and later

The `ProxyFtpEscapeWildcards` directive controls whether wildcard characters ("*?[]~") in requested filenames are escaped with backslash before sending them to the FTP server. That is the default behavior, but many FTP servers don't know about the escaping and try to serve the literal filenames they were sent, including the backslashes in the names.

Set to "off" to allow downloading files with wildcards in their names from FTP servers that don't understand wildcard escaping.
| **Description:** | Whether wildcards in requested filenames trigger a file listing |
| **Syntax:** | `ProxyFtpListOnWildcard [on|off]` |
| **Default:** | on |
| **Context:** | server config, virtual host, directory |
| **Status:** | Extension |
| **Module:** | `mod_proxy_ftp` |
| **Compatibility:** | Available in Apache 2.3.3 and later |

The `ProxyFtpListOnWildcard` directive controls whether wildcard characters (`"*?[~"]`) in requested filenames cause `mod_proxy_ftp` to return a listing of files instead of downloading a file. By default (value on), they do. Set to "off" to allow downloading files even if they have wildcard characters in their names.
Apache Module mod_proxy_hcheck

**Description:** Dynamic health check of Balancer members (workers) for mod_proxy

**Status:** Extension

**Module Identifier:** proxy_hcheck_module

**Source File:** mod_proxy_hcheck.c

**Compatibility:** Available in Apache 2.4.21 and later

**Summary**

This module provides for dynamic health checking of balancer members (workers). This can be enabled on a worker-by-worker basis. The health check is done independently of the actual reverse proxy requests.

This module *requires* the service of mod_watchdog.

**Parameters**

The health check mechanism is enabled via the use of additional BalancerMember parameters, which are configured in the standard way via `ProxyPass`:

A new BalancerMember status state (flag) is defined via this module: "C". When the worker is taken offline due to failures as determined by the health check module, this flag is set, and can be seen (and modified) via the balancer-manager.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hcmethod</td>
<td>None</td>
<td>No dynamic health check performed. Choices are:</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No dynamic health</td>
<td></td>
</tr>
<tr>
<td>Checking</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><strong>TCP</strong></td>
<td>Check that a socket to the backend can be created: e.g. &quot;are you up&quot;</td>
<td></td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Send an HTTP OPTIONS request to the backend</td>
<td></td>
</tr>
<tr>
<td>HEAD</td>
<td>Send an HTTP HEAD request to the backend</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>Send an HTTP GET request to the backend</td>
<td></td>
</tr>
</tbody>
</table>

*: Unless hcexpr is used, a 2xx or 3xx HTTP status will be interpreted as _passing_ the health check

| hcpasses | 1 | Number of successful health check tests before worker is re-enabled |
| hcfails | 1 | Number of failed health check tests before worker is disabled |
| hcinterval | 30 | Period of health checks in seconds (e.g. performed every 30 seconds) |
| hcuri | | Additional URI to be appended to the worker URL for the health check. |
| hctemplate | | Name of template, created via `ProxyHCTemplate` to use for setting health check parameters for this worker |
| hcexpr | | Name of expression, created via `ProxyHCEExpr`, used to check response headers for health. |

_If not used, 2xx thru 3xx status codes imply success_
Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod_proxy
The following example shows how one might configure health checking for various backend servers:

```plaintext
ProxyHCExpr ok234 {%{REQUEST_STATUS} =~ /^[234]/}
ProxyHCExpr gdown {%{REQUEST_STATUS} =~ /^[5]/}
ProxyHCExpr in_maint {hc('body') !~ /Under maintenance/}

<Proxy balancer://foo>
    BalancerMember http://www.example.com/ hcmethod=GET hcexpr=in_maint curi=/status.php
    BalancerMember http://www2.example.com/ hcmethod=HEAD hcexpr=ok234 hcinterval=10
    BalancerMember http://www3.example.com/ hcmethod=TCP hcinterval=5 hcpasses=2 hcfails=3
    BalancerMember http://www4.example.com/
</Proxy>

ProxyPass "/" "balancer://foo"
ProxyPassReverse "/" "balancer://foo"
```

In this scenario, `http://www.example.com/` is health checked by sending a `GET /status.php` request to that server and seeing that the returned page does not include the string `Under maintenance`. If it does, that server is put in health-check fail mode, and disabled. This dynamic check is performed every 30 seconds, which is the default.

`http://www2.example.com/` is checked by sending a simple `HEAD` request every 10 seconds and making sure that the response status is 2xx, 3xx or 4xx.

`http://www3.example.com/` is checked every 5 seconds by simply ensuring that the socket to that server is up. If the backend is marked as "down" and it passes 2 health check, it will be re-enabled and added back into the load balancer. It takes 3 back-to-back health check failures to disable the server and move it out of rotation. Finally, `http://www4.example.com/` is not
dynamically checked at all.
**Description:** Creates a named condition expression to use to determine health of the backend based on its response.

**Syntax:**
```
ProxyHCExpr name {ap_expr expression}
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_proxy_hcheck

The `ProxyHCExpr` directive allows for creating a named condition expression that checks the response headers of the backend server to determine its health. This named condition can then be assigned to balancer members via the `hcexpr` parameter.

### ProxyHCExpr: Allow for 2xx/3xx/4xx as passing

```
ProxyHCExpr ok234 {%{REQUEST_STATUS} =~ /[234]/}
ProxyPass "/apps" "http://backend.example.com/" hcexpr=ok234
```

The **expression** can use curly-parens (`"{}"`) as quoting delimiters in addition to normal quotes.

If using a health check method (eg: GET) which results in a response body, that body itself can be checked via `ap_expr` using the `hc()` expression function, which is unique to this module.

In the following example, we send the backend a GET request and if the response body contains the phrase *Under maintenance*, we want to disable the backend.

### ProxyHCExpr: Checking response body

```
ProxyHCExpr in_maint {hc('body') !~ /Under maintenance/}
ProxyPass "/apps" "http://backend.example.com/" hcexpr=in_maint
```

NOTE: Since response body can quite large, it is best if used against specific status pages.
**Description:** Creates a named template for setting various health check parameters

**Syntax:**
```
ProxyHCTemplate name
parameter=setting <...>
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_proxy_hcheck

The *ProxyHCTemplate* directive allows for creating a named set (template) of health check parameters that can then be assigned to balancer members via the hctemplate parameter.

---

**ProxyHCTemplate**

```
ProxyHCTemplate tcp5 hcmethod=tcp hcinterval=5
ProxyPass "/apps" "http://backend.example.com/" hctemplate=tcp5
```
**Description:** Sets the total server-wide size of the threadpool used for the health check workers.

**Syntax:**

```
ProxyHCTPsize <size>
```

**Context:** server config

**Status:** Extension

**Module:** mod_proxy_hcheck

If Apache httpd and APR are built with thread support, the health check module will offload the work of the actual checking to a threadpool associated with the Watchdog process, allowing for parallel checks. The `ProxyHCTPsize` directive determines the size of this threadpool. If set to 0, no threadpool is used at all, resulting in serialized health checks. The default size is 16.

**ProxyHCTPsize**

```
ProxyHCTPsize 32
```
Apache Module mod_proxy_html

| Description: | Rewrite HTML links in to ensure they are addressable from Clients' networks in a proxy context. |
| Status: | Base |
| Module Identifier: | proxy_html_module |
| Source File: | mod_proxy_html.c |
| Compatibility: | Version 2.4 and later. Available as a third-party module for earlier 2.x versions |

Summary

This module provides an output filter to rewrite HTML links in a proxy situation, to ensure that links work for users outside the proxy. It serves the same purpose as Apache's ProxyPassReverse directive does for HTTP headers, and is an essential component of a reverse proxy.

For example, if a company has an application server at appserver.example.com that is only visible from within the company's internal network, and a public webserver www.example.com, they may wish to provide a gateway to the application server at http://www.example.com/appserver/. When the application server links to itself, those links need to be rewritten to work through the gateway. mod_proxy_html serves to rewrite `<a href="http://appserver.example.com/foo/bar.html">foobar</a>` to `<a href="http://www.example.com/appserver/foo/bar.html">foobar</a>`, making it accessible from outside.

mod_proxy_html was originally developed at WebPing, whose extensive documentation may be useful to users.
### Description
Sets the buffer size increment for buffering inline scripts and stylesheets.

### Syntax
ProxyHTMLBufSize bytes

### Context
server config, virtual host, directory

### Status
Base

### Module
mod_proxy_html

### Compatibility
Version 2.4 and later; available as a third-party for earlier 2.x versions

In order to parse non-HTML content (stylesheets and scripts) embedded in HTML documents, mod_proxy_html has to read the entire script or stylesheet into a buffer. This buffer will be expanded as necessary to hold the largest script or stylesheet in a page, in increments of bytes as set by this directive.

The default is 8192, and will work well for almost all pages. However, if you know you're proxying pages containing stylesheets and/or scripts bigger than 8K (that is, for a single script or stylesheet, NOT in total), it will be more efficient to set a larger buffer size and avoid the need to resize the buffer dynamically during a request.
**Description:** Specify a charset for mod_proxy_html output.

**Syntax:**
ProxyHTMLCharsetOut `Charset` | `*`

**Context:**
server config, virtual host, directory

**Status:**
Base

**Module:**
mod_proxy_html

**Compatibility:**
Version 2.4 and later; available as a third-party for earlier 2.x versions

This selects an encoding for mod_proxy_html output. It should not normally be used, as any change from the default UTF-8 (Unicode - as used internally by libxml2) will impose an additional processing overhead. The special token `ProxyHTMLCharsetOut *` will generate output using the same encoding as the input.

Note that this relies on `mod_xml2enc` being loaded.
### ProxyHTMLDocType

**Description:** Sets an HTML or XHTML document type declaration.

**Syntax:**
- `ProxyHTMLDocType HTML | XHTML [Legacy]`
- `ProxyHTMLDocType fpi [SGML | XML]`

**Context:** server config, virtual host, directory

**Status:** Base

**Module:** mod_proxy_html

**Compatibility:** Version 2.4 and later; available as a third-party for earlier 2.x versions

---

In the first form, documents will be declared as HTML 4.01 or XHTML 1.0 according to the option selected. This option also determines whether HTML or XHTML syntax is used for output. Note that the format of the documents coming from the backend server is immaterial: the parser will deal with it automatically. If the optional second argument is set to "Legacy", documents will be declared "Transitional", an option that may be necessary if you are proxying pre-1998 content or working with defective authoring/publishing tools.

In the second form, it will insert your own FPI. The optional second argument determines whether SGML/HTML or XML/XHTML syntax will be used.

The default is changed to omitting any FPI, on the grounds that no FPI is better than a bogus one. If your backend generates decent HTML or XHTML, set it accordingly.

If the first form is used, mod_proxy_html will also clean up the HTML to the specified standard. It cannot fix every error, but it will strip out bogus elements and attributes. It will also optionally log other errors at [LogLevel Debug](#)
**Description:** Turns the proxy_html filter on or off.

**Syntax:**

ProxyHTMLEnable 0n|0ff

**Default:**

ProxyHTMLEnable Off

**Context:** server config, virtual host, directory

**Status:** Base

**Module:** mod_proxy_html

**Compatibility:** Version 2.4 and later; available as a third-party module for earlier 2.x versions.

A simple switch to enable or disable the proxy_html filter. If *mod_xml2enc* is loaded it will also automatically set up internationalisation support.

Note that the proxy_html filter will only act on HTML data (Content-Type text/html or application/xhtml+xml) and when the data are proxied. You can override this (at your own risk) by setting the `PROXY_HTML_FORCE` environment variable.
### ProxyHTMLEvents

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Specify attributes to treat as scripting events.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>ProxyHTMLEvents <em>attribute</em> [ <em>attribute</em> ... ]</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_proxy_html</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Version 2.4 and later; available as a third-party for earlier 2.x versions</td>
</tr>
</tbody>
</table>

Specifies one or more attributes to treat as scripting events and apply **ProxyHTMLURLMaps** to where enabled. You can specify any number of attributes in one or more **ProxyHTMLEvents** directives.

Normally you'll set this globally. If you set **ProxyHTMLEvents** in more than one scope so that one overrides the other, you'll need to specify a complete set in each of those scopes.

A default configuration is supplied in *proxy-html.conf* and defines the events in standard HTML 4 and XHTML 1.
**Description:** Determines whether to fix links in inline scripts, stylesheets, and scripting events.

**Syntax:** `ProxyHTMLExtended On|Off`

**Default:** `ProxyHTMLExtended Off`

**Context:** server config, virtual host, directory

**Status:** Base

**Module:** `mod_proxy_html`

**Compatibility:** Version 2.4 and later; available as a third-party for earlier 2.x versions

---

Set to **Off**, HTML links are rewritten according to the `ProxyHTMLURLMap` directives, but links appearing in Javascript and CSS are ignored.

Set to **On**, all scripting events (as determined by `ProxyHTMLEvents`) and embedded scripts or stylesheets are also processed by the `ProxyHTMLURLMap` rules, according to the flags set for each rule. Since this requires more parsing, performance will be best if you only enable it when strictly necessary.

You'll also need to take care over patterns matched, since the parser has no knowledge of what is a URL within an embedded script or stylesheet. In particular, extended matching of `/` is likely to lead to false matches.
| **Description:** | Fixes for simple HTML errors. |
| **Syntax:** | ProxyHTMLFixups *[lowercase]*  
*[dospath]* *[reset]* |
| **Context:** | server config, virtual host, directory |
| **Status:** | Base |
| **Module:** | mod_proxy_html |
| **Compatibility:** | Version 2.4 and later; available as a third-party for earlier 2.x versions |

This directive takes one to three arguments as follows:

- **lowercase** Uris are rewritten to lowercase
- **dospath** Backslashes in URLs are rewritten to forward slashes.
- **reset** Unset any options set at a higher level in the configuration.

Take care when using these. The fixes will correct certain authoring mistakes, but risk also erroneously fixing links that were correct to start with. Only use them if you know you have a broken backend server.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Enables per-request interpolation of ProxyHTMLURLMap rules.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>ProxyHTMLInterp <em>On</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>ProxyHTMLInterp <em>Off</em></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_proxy_html</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Version 2.4 and later; available as a third-party for earlier 2.x versions</td>
</tr>
</tbody>
</table>

This enables per-request interpolation in ProxyHTMLURLMap to-and from- patterns.

If interpolation is not enabled, all rules are pre-compiled at startup. With interpolation, they must be re-compiled for every request, which implies an extra processing overhead. It should therefore be enabled only when necessary.
## Description
Specify HTML elements that have URL attributes to be rewritten.

## Syntax
ProxyHTMLLinks element attribute [attribute2 ...]

## Context
server config, virtual host, directory

## Status
Base

## Module
mod_proxy_html

## Compatibility
Version 2.4 and later; available as a third-party for earlier 2.x versions

Specifies elements that have URL attributes that should be rewritten using standard ProxyHTMLURLMaps. You will need one ProxyHTMLLinks directive per element, but it can have any number of attributes.

Normally you'll set this globally. If you set ProxyHTMLLinks in more than one scope so that one overrides the other, you'll need to specify a complete set in each of those scopes.

A default configuration is supplied in *proxy-html.conf* and defines the HTML links for standard HTML 4 and XHTML 1.

### Examples from proxy-html.conf

<table>
<thead>
<tr>
<th>ProxyHTMLLinks</th>
<th>a</th>
<th>href</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProxyHTMLLinks</td>
<td>area</td>
<td>href</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>link</td>
<td>href</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>img</td>
<td>src</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>object</td>
<td>classid codebase data usemap</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>q</td>
<td>cite</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>blockquote</td>
<td>cite</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>ins</td>
<td>cite</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>del</td>
<td>cite</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>form</td>
<td>action</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>input</td>
<td>src usemap</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>head</td>
<td>profile</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>base</td>
<td>href</td>
</tr>
<tr>
<td>ProxyHTMLLinks</td>
<td>script</td>
<td>src for</td>
</tr>
</tbody>
</table>
**Description:** Turns on or off extra pre-parsing of metadata in HTML `<head>` sections.

**Syntax:** ProxyHTMLMeta On|Off

**Default:** ProxyHTMLMeta Off

**Context:** server config, virtual host, directory

**Status:** Base

**Module:** mod_proxy_html

**Compatibility:** Version 2.4 and later; available as a third-party module for earlier 2.x versions.

This turns on or off pre-parsing of metadata in HTML `<head>` sections.

If not required, turning ProxyHTMLMeta Off will give a small performance boost by skipping this parse step. However, it is sometimes necessary for internationalisation to work correctly.

ProxyHTMLMeta has two effects. Firstly and most importantly it enables detection of character encodings declared in the form `<meta http-equiv="Content-Type" content="text/html">

or, in the case of an XHTML document, an XML declaration. It is NOT required if the charset is declared in a real HTTP header (which is always preferable) from the backend server, nor if the document is utf-8 (unicode) or a subset such as ASCII. You may also be able to dispense with it where documents use a default declared using `xml2EncDefault`, but that risks propagating an incorrect declaration. A `ProxyHTMLCharsetOut` can remove that risk, but is likely to be a bigger processing overhead than enabling ProxyHTMLMeta.

The other effect of enabling ProxyHTMLMeta is to parse all `<meta>`
http-equiv=...> declarations and convert them to real HTTP headers, in keeping with the original purpose of this form of the HTML <meta> element.

**Warning**

Because ProxyHTMLMeta promotes all `http-equiv` elements to HTTP headers, it is important that you only enable it in cases where you trust the HTML content as much as you trust the upstream server. If the HTML is controlled by bad actors, it will be possible for them to inject arbitrary, possibly malicious, HTTP headers into your server's responses.
**Description:** Determines whether to strip HTML comments.

**Syntax:**
```
ProxyHTMLStripComments On|Off
```

**Default:**
```
ProxyHTMLStripComments Off
```

**Context:** server config, virtual host, directory

**Status:** Base

**Module:** mod_proxy_html

**Compatibility:** Version 2.4 and later; available as a third-party for earlier 2.x versions

This directive will cause mod_proxy_html to strip HTML comments. Note that this will also kill off any scripts or styles embedded in comments (a bogosity introduced in 1995/6 with Netscape 2 for the benefit of then-older browsers, but still in use today). It may also interfere with comment-based processors such as SSI or ESI: be sure to run any of those before mod_proxy_html in the filter chain if stripping comments!
**Description:** Defines a rule to rewrite HTML links

**Syntax:**
ProxyHTMLURLMap *from-pattern* *to-pattern* [flags] [cond]

**Context:**
server config, virtual host, directory

**Status:**
Base

**Module:**
mod_proxy_html

**Compatibility:**
Version 2.4 and later; available as a third-party module for earlier 2.x versions.

This is the key directive for rewriting HTML links. When parsing a document, whenever a link target matches *from-pattern*, the matching portion will be rewritten to *to-pattern*, as modified by any flags supplied and by the `ProxyHTMLExtended` directive. Only the elements specified using the `ProxyHTMLLinks` directive will be considered as HTML links.

The optional third argument may define any of the following **Flags**. Flags are case-sensitive.

- **h**
  Ignore HTML links (pass through unchanged)

- **e**
  Ignore scripting events (pass through unchanged)

- **c**
  Pass embedded script and style sections through untouched.

- **L**
  Last-match. If this rule matches, no more rules are applied (note that this happens automatically for HTML links).

- **l**
  Opposite to L. Overrides the one-change-only default
behaviour with HTML links.

R

Use Regular Expression matching-and-replace. from-pattern is a regexp, and to-pattern a replacement string that may be based on the regexp. Regexp memory is supported: you can use brackets () in the from-pattern and retrieve the matches with $1 to $9 in the to-pattern.

If R is not set, it will use string-literal search-and-replace. The logic is starts-with in HTML links, but contains in scripting events and embedded script and style sections.

X

Use POSIX extended Regular Expressions. Only applicable with R.

i

Case-insensitive matching. Only applicable with R.

n

Disable regexp memory (for speed). Only applicable with R.

s

Line-based regexp matching. Only applicable with R.

^ Match at start only. This applies only to string matching (not regexps) and is irrelevant to HTML links.

$ Match at end only. This applies only to string matching (not regexps) and is irrelevant to HTML links.

V Interpolate environment variables in to-pattern. A string of
the form ${varname|default} will be replaced by the value of environment variable varname. If that is unset, it is replaced by default. The |default is optional.

NOTE: interpolation will only be enabled if ProxyHTMLInterp is On.

v

Interpolate environment variables in from-pattern. Patterns supported are as above.

NOTE: interpolation will only be enabled if ProxyHTMLInterp is On.

The optional fourth cond argument defines a condition that will be evaluated per Request, provided ProxyHTMLInterp is On. If the condition evaluates FALSE the map will not be applied in this request. If TRUE, or if no condition is defined, the map is applied.

A cond is evaluated by the Expression Parser. In addition, the simpler syntax of conditions in mod_proxy_html 3.x for HTTPD 2.0 and 2.2 is also supported.
Apache Module mod_proxy_http

| Description: | HTTP support module for mod_proxy |
| Status:      | Extension                        |
| Module Identifier: | proxy_http_module                |
| Source File: | mod_proxy_http.c                  |

Summary

This module requires the service of mod_proxy. It provides the features used for proxying HTTP and HTTPS requests. mod_proxy_http supports HTTP/0.9, HTTP/1.0 and HTTP/1.1. It does not provide any caching abilities. If you want to set up a caching proxy, you might want to use the additional service of the mod_cache module.

Thus, in order to get the ability of handling HTTP proxy requests, mod_proxy and mod_proxy_http have to be present in the server.

Warning

Do not enable proxying until you have secured your server. Open proxy servers are dangerous both to your network and to the Internet at large.

Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod_proxy
mod_proxy_connect
In addition to the configuration directives that control the behaviour of `mod_proxy`, there are a number of `environment variables` that control the HTTP protocol provider. Environment variables below that don't specify specific values are enabled when set to any value.

**proxy-sendextracrlf**
Causes proxy to send an extra CR-LF newline on the end of a request. This is a workaround for a bug in some browsers.

**force-proxy-request-1.0**
Forces the proxy to send requests to the backend as HTTP/1.0 and disables HTTP/1.1 features.

**proxy-nokeepalive**
Forces the proxy to close the backend connection after each request.

**proxy-chain-auth**
If the proxy requires authentication, it will read and consume the proxy authentication credentials sent by the client. With `proxy-chain-auth` it will also forward the credentials to the next proxy in the chain. This may be necessary if you have a chain of proxies that share authentication information. **Security Warning:** Do not set this unless you know you need it, as it forwards sensitive information!

**proxy-sendcl**
HTTP/1.0 required all HTTP requests that include a body (e.g. POST requests) to include a `Content-Length` header. This environment variable forces the Apache proxy to send this header to the backend server, regardless of what the Client sent to the proxy. It ensures compatibility when proxying for an HTTP/1.0 or unknown backend. However, it may require the entire request to be buffered by the proxy, so it becomes very inefficient for large requests.
**proxy-sendchunks or proxy-sendchunked**
This is the opposite of *proxy-sendcl*. It allows request bodies to be sent to the backend using chunked transfer encoding. This allows the request to be efficiently streamed, but requires that the backend server supports HTTP/1.1.

**proxy-interim-response**
This variable takes values RFC (the default) or Suppress. Earlier httpd versions would suppress HTTP interim (1xx) responses sent from the backend. This is technically a violation of the HTTP protocol. In practice, if a backend sends an interim response, it may itself be extending the protocol in a manner we know nothing about, or just broken. So this is now configurable: set `proxy-interim-response` RFC to be fully protocol compliant, or `proxy-interim-response` Suppress to suppress interim responses.

**proxy-initial-not-pooled**
If this variable is set, no pooled connection will be reused if the client request is the initial request on the frontend connection. This avoids the "proxy: error reading status line from remote server" error message caused by the race condition that the backend server closed the pooled connection after the connection check by the proxy and before data sent by the proxy reached the backend. It has to be kept in mind that setting this variable downgrades performance, especially with HTTP/1.0 clients.
mod_proxy_http creates the following request notes for logging using the \%{VARNAME}n format in LogFormat or ErrorLogFormat:

**proxy-source-port**

The local port used for the connection to the backend server.

**proxy-status**

The HTTP status received from the backend server.
Apache Module mod_proxy_http2

**Description:** HTTP/2 support module for mod_proxy

**Status:** Extension

**Module Identifier:** proxy_http2_module

**Source File:** mod_proxy_http2.c

**Summary**

*mod_proxy_http2* supports HTTP/2 only, it does *not* provide any downgrades to HTTP/1.1. This means that the backend needs to support HTTP/2 because HTTP/1.1 will not be used instead.

This module *requires* the service of *mod_proxy*, so in order to get the ability of handling HTTP/2 proxy requests, *mod_proxy* and *mod_proxy_http2* need to be both loaded by the server.

*mod_proxy_http2* works with incoming fronted requests using HTTP/1.1 or HTTP/2. In both cases, requests proxied to the same backend are sent over a single TCP connection whenever possible (namely when the connection can be re-used).

Caveat: there will be no attempt to consolidate multiple HTTP/1.1 frontend requests (configured to be proxied to the same backend) into HTTP/2 streams belonging to the same HTTP/2 request. Each HTTP/1.1 frontend request will be proxied to the backend using a separate HTTP/2 request (trying to re-use the same TCP connection if possible).

This module relies on *libnghttp2* to provide the core http/2 engine.

**Warning**

This module is experimental. Its behaviors, directives, and defaults are subject to more change from release to release relative to other
standard modules. Users are encouraged to consult the "CHANGES" file for potential updates.

Warning
Do not enable proxying until you have secured your server. Open proxy servers are dangerous both to your network and to the Internet at large.

Bugfix checklist
httpd changelog
Known issues
Report a bug

See also
mod_http2
mod_proxy
mod_proxy_connect
The examples below demonstrate how to configure HTTP/2 for backend connections for a reverse proxy.

### HTTP/2 (TLS)

<table>
<thead>
<tr>
<th>Command</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProxyPass</td>
<td>/app &quot;h2://app.example.com&quot;</td>
</tr>
<tr>
<td>ProxyPassReverse</td>
<td>/app &quot;<a href="https://app.example.com">https://app.example.com</a>&quot;</td>
</tr>
</tbody>
</table>

### HTTP/2 (cleartext)

<table>
<thead>
<tr>
<th>Command</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProxyPass</td>
<td>/app &quot;h2c://app.example.com&quot;</td>
</tr>
<tr>
<td>ProxyPassReverse</td>
<td>/app &quot;<a href="http://app.example.com">http://app.example.com</a>&quot;</td>
</tr>
</tbody>
</table>

The schemes to configure above in `ProxyPassReverse` for reverse proxying h2 (or h2c) protocols are the usual `https` (resp. `http`) as expected/used by the user agent.
mod_proxy_http creates the following request notes for logging using the \%{VARNAME}n format in LogFormat or ErrorLogFormat:

**proxy-source-port**
- The local port used for the connection to the backend server.

**proxy-status**
- The HTTP/2 status received from the backend server.
Apache Module mod_proxy_scgi

Description: SCGI gateway module for mod_proxy
Status: Extension
Module Identifier: proxy_scgi_module
Source File: mod_proxy_scgi.c
Compatibility: Available in version 2.2.14 and later

Summary
This module requires the service of mod_proxy. It provides support for the SCGI protocol, version 1.

Thus, in order to get the ability of handling the SCGI protocol, mod_proxy and mod_proxy_scgi have to be present in the server.

Warning
Do not enable proxying until you have secured your server. Open proxy servers are dangerous both to your network and to the Internet at large.

Bugfix checklist
httpd changelog
Known issues
Report a bug

See also
mod_proxy
mod_proxy_balancer
Remember, in order to make the following examples work, you have to enable `mod_proxy` and `mod_proxy_scgi`.

**Simple gateway**

```
ProxyPass /scgi-bin/ scgi://localhost:4000/
```

The balanced gateway needs `mod_proxy_balancer` and at least one load balancer algorithm module, such as `mod_lbmetho_byrequests`, in addition to the proxy modules listed above. `mod_lbmetho_byrequests` is the default, and will be used for this example configuration.

**Balanced gateway**

```
ProxyPass "/scgi-bin/" "balancer://somecluster/"
<Proxy "balancer://somecluster">
    BalancerMember "scgi://localhost:4000"
    BalancerMember "scgi://localhost:4001"
</Proxy>
```
In addition to the configuration directives that control the behaviour of `mod_proxy`, an environment variable may also control the SCGI protocol provider:

**proxy-scgi-pathinfo**

By default `mod_proxy_scgi` will neither create nor export the `PATH_INFO` environment variable. This allows the backend SCGI server to correctly determine `SCRIPT_NAME` and `Script-URI` and be compliant with RFC 3875 section 3.3. If instead you need `mod_proxy_scgi` to generate a "best guess" for `PATH_INFO`, set this env-var. The variable must be set before `SetEnv` is effective. `SetEnvIf` can be used instead: `SetEnvIf Request_URI . proxy-scgi-pathinfo`
**Description:** Enable or disable internal redirect responses from the backend

**Syntax:** `ProxySCGIInternalRedirect On|Off|Headername`

**Default:** `ProxySCGIInternalRedirect On`

**Context:** server config, virtual host, directory

**Status:** Extension

**Module:** `mod_proxy_scgi`

**Compatibility:** The `Headername` feature is available in version 2.4.13 and later

The `ProxySCGIInternalRedirect` enables the backend to internally redirect the gateway to a different URL. This feature originates in `mod_cgi`, which internally redirects the response if the response status is OK (200) and the response contains a Location (or configured alternate header) and its value starts with a slash (`/`). This value is interpreted as a new local URL that Apache httpd internally redirects to.

`mod_proxy_scgi` does the same as `mod_cgi` in this regard, except that you can turn off the feature or specify the use of a header other than Location.

**Example**

```
ProxySCGIInternalRedirect Off

# Django and some other frameworks will fully qualify "local URI
# set by the application, so an alternate header must be used.
<Location /django-app/>
    ProxySCGIInternalRedirect X-Location
</Location>
```
**Description:** Enable evaluation of *X-Sendfile* pseudo response header

**Syntax:**

```
ProxySCGISendfile On|Off|Headername
```

**Default:**

```
ProxySCGISendfile Off
```

**Context:** server config, virtual host, directory

**Status:** Extension

**Module:** `mod_proxy_scgi`

---

The `ProxySCGISendfile` directive enables the SCGI backend to let files be served directly by the gateway. This is useful for performance purposes — `httpd` can use `sendfile` or other optimizations, which are not possible if the file comes over the backend socket. Additionally, the file contents are not transmitted twice.

The `ProxySCGISendfile` argument determines the gateway behaviour:

**Off**

No special handling takes place.

**On**

The gateway looks for a backend response header called *X-Sendfile* and interprets the value as the filename to serve. The header is removed from the final response headers. This is equivalent to `ProxySCGISendfile X-Sendfile`.

**anything else**

Similar to **On**, but instead of the hardcoded header name *X-Sendfile*, the argument is used as the header name.

---

**Example**

```
# Use the default header (X-Sendfile)
ProxySCGISendfile On
```
# Use a different header
ProxySCGISendfile X-Send-Static
Apache Module mod_proxy_wstunnel

**Description:** Websockets support module for mod_proxy

**Status:** Extension

**Module Identifier:** proxy_wstunnel_module

**Source File:** mod_proxy_wstunnel.c

**Compatibility:** Available in httpd 2.4.5 and later

**Summary**

This module *requires* the service of mod_proxy. It provides support for the tunnelling of web socket connections to a backend websockets server. The connection is automatically upgraded to a websocket connection:

**HTTP Response**

Upgrade: WebSocket  
Connection: Upgrade

Proxying requests to a websockets server like echo.websocket.org can be done using the **ProxyPass** directive:

```
ProxyPass "/ws2/" "ws://echo.websocket.org/"
ProxyPass "/wss2/" "wss://echo.websocket.org/"
```

Load balancing for multiple backends can be achieved using **mod_proxy_balancer**.

In fact the module can be used to upgrade to other protocols, you can set the upgrade parameter in the **ProxyPass** directive to allow the module to accept other protocol. NONE means you bypass the check for the header but still upgrade to WebSocket. ANY means that
Upgrade will read in the request headers and use in the response

Upgrade

Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod proxy
# Apache Module mod_ratelimit

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Bandwidth Rate Limiting for Clients</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>ratelimit_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_ratelimit.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>rate-initial-burst available in httpd 2.4.24 and later.</td>
</tr>
</tbody>
</table>

## Summary

Provides a filter named RATE_LIMIT to limit client bandwidth. The throttling is applied to each HTTP response while it is transferred to the client, and not aggregated at IP/client level. The connection speed to be simulated is specified, in KiB/s, using the environment variable rate-limit.

Optionally, an initial amount of burst data, in KiB, may be configured to be passed at full speed before throttling to the specified rate limit. This value is optional, and is set using the environment variable rate-initial-burst.

## Example Configuration

```xml
<Location "/downloads">
  SetOutputFilter RATE_LIMIT
  SetEnv rate-limit 400
  SetEnv rate-initial-burst 512
</Location>
```

If the value specified for rate-limit causes integer overflow, the rate-limited will be disabled. If the value specified for rate-limit-burst causes integer overflow, the burst will be disabled.
Apache Module mod_reflector

**Description:** Reflect a request body as a response via the output filter stack.

**Status:** Base

**Module Identifier:** reflector_module

**Source File:** mod_reflector.c

**Compatibility:** Version 2.3 and later

**Summary**

This module allows request bodies to be reflected back to the client, in the process passing the request through the output filter stack. A suitably configured chain of filters can be used to transform the request into a response. This module can be used to turn an output filter into an HTTP service.
Compression service
   Pass the request body through the DEFLATE filter to compress the body. This request requires a Content-Encoding request header containing "gzip" for the filter to return compressed data.

   <Location "/compress">
      SetHandler reflector
      SetOutputFilter DEFLATE
   </Location>

Image downsampling service
   Pass the request body through an image downsampling filter, and reflect the results to the caller.

   <Location "/downsample">
      SetHandler reflector
      SetOutputFilter DOWNSAMPLE
   </Location>
**Description:** Reflect an input header to the output headers

**Syntax:**
```
ReflectorHeader inputheader [outputheader]
```

**Context:** server config, virtual host, directory, .htaccess

**Override:** Options

**Status:** Base

**Module:** mod_reflector

This directive controls the reflection of request headers to the response. The first argument is the name of the request header to copy. If the optional second argument is specified, it will be used as the name of the response header, otherwise the original request header name will be used.
Apache Module mod_remoteip

**Description:**
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.

**Status:**
Base

**Module Identifier:**
remoteip_module

**Source File:**
mod_remoteip.c

**Summary**

This module is used to treat the useragent which initiated the request as the originating useragent as identified by httpd for the purposes of authorization and logging, even where that useragent is behind a load balancer, front end server, or proxy server.

The module overrides the client IP address for the connection with the useragent IP address reported in the request header configured with the `RemoteIPHeader` directive.

Once replaced as instructed, this overridden useragent IP address is then used for the `mod_authz_host Require ip` feature, is reported by `mod_status`, and is recorded by `mod_log_config %a` and `core %a` format strings. The underlying client IP of the connection is available in the `%{c}a` format string.

It is critical to only enable this behavior from intermediate hosts (proxies, etc) which are trusted by this server, since it is trivial for the remote useragent to impersonate another useragent.
Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod_authz_host
mod_status
mod_log_config
Apache by default identifies the useragent with the connection's client_ip value, and the connection remote_host and remote_logname are derived from this value. These fields play a role in authentication, authorization and logging and other purposes by other loadable modules.

mod_remoteip overrides the client IP of the connection with the advertised useragent IP as provided by a proxy or load balancer, for the duration of the request. A load balancer might establish a long lived keepalive connection with the server, and each request will have the correct useragent IP, even though the underlying client IP address of the load balancer remains unchanged.

When multiple, comma delimited useragent IP addresses are listed in the header value, they are processed in Right-to-Left order. Processing halts when a given useragent IP address is not trusted to present the preceding IP address. The header field is updated to this remaining list of unconfirmed IP addresses, or if all IP addresses were trusted, this header is removed from the request altogether.

In overriding the client IP, the module stores the list of intermediate hosts in a remoteip-proxy-ip-list note, which mod_log_config can record using the %{remoteip-proxy-ip-list}n format token. If the administrator needs to store this as an additional header, this same value can also be recording as a header using the directive RemoteIPProxiesHeader.

IPv4-over-IPv6 Mapped Addresses
As with httpd in general, any IPv4-over-IPv6 mapped addresses are recorded in their IPv4 representation.

Internal (Private) Addresses
All internal addresses 10/8, 172.16/12, 192.168/16, 169.254/16 and 127/8 blocks (and IPv6 addresses outside of the public 2000::/3 block) are only evaluated by mod_remoteip when `RemoteIPInternalProxy` internal (intranet) proxies are registered.
**Description:** Declare the header field which should be parsed for useragent IP addresses

**Syntax:** RemoteIPHeader *header-field*

**Context:** server config, virtual host

**Status:** Base

**Module:** mod_remoteip

The **RemoteIPHeader** directive triggers `mod_remoteip` to treat the value of the specified *header-field* header as the useragent IP address, or list of intermediate useragent IP addresses, subject to further configuration of the **RemoteIPInternalProxy** and **RemoteIPTrustedProxy** directives. Unless these other directives are used, `mod_remoteip` will trust all hosts presenting a **RemoteIPHeader** IP value.

### Internal (Load Balancer) Example

RemoteIPHeader X-Client-IP

### Proxy Example

RemoteIPHeader X-Forwarded-For
The `RemoteIPInternalProxy` directive adds one or more addresses (or address blocks) to trust as presenting a valid RemoteIPHeader value of the useragent IP. Unlike the `RemoteIPTrustedProxy` directive, any IP address presented in this header, including private intranet addresses, are trusted when passed from these proxies.

### Internal (Load Balancer) Example

```plaintext
RemoteIPHeader X-Client-IP
RemoteIPInternalProxy 10.0.2.0/24
RemoteIPInternalProxy gateway.localdomain
```
**Description:** Declare client intranet IP addresses trusted to present the RemoteIPHeader value

**Syntax:** RemoteIPInternalProxyList *filename*

**Context:** server config, virtual host

**Status:** Base

**Module:** mod_remoteip

The `RemoteIPInternalProxyList` directive specifies a file parsed at startup, and builds a list of addresses (or address blocks) to trust as presenting a valid RemoteIPHeader value of the useragent IP.

The '#' hash character designates a comment line, otherwise each whitespace or newline separated entry is processed identically to the `RemoteIPInternalProxy` directive.

**Internal (Load Balancer) Example**

RemoteIPHeader X-Client-IP
RemoteIPInternalProxyList conf/trusted-proxies.lst

**conf/trusted-proxies.lst contents**

```
# Our internally trusted proxies;
10.0.2.0/24 #Everyone in the testing group
gateway.localdomain #The front end balancer
```
**Description:** Declare the header field which will record all intermediate IP addresses

**Syntax:** RemoteIPProxiesHeader *HeaderFieldName*

**Context:** server config, virtual host

**Status:** Base

**Module:** mod_remoteip

The `RemoteIPProxiesHeader` directive specifies a header into which `mod远程ip` will collect a list of all of the intermediate client IP addresses trusted to resolve the useragent IP of the request. Note that intermediate `RemoteIPTrustedProxy` addresses are recorded in this header, while any intermediate `RemoteIPInternalProxy` addresses are discarded.

**Example**

```
RemoteIPHeader X-Forwarded-For
RemoteIPProxiesHeader X-Forwarded-By
```
**RemoteIPTrustedProxy Directive**

**Description:** Declare client intranet IP addresses trusted to present the RemoteIPHeader value

**Syntax:** RemoteIPTrustedProxy `proxy-ip|proxy-ip/subnet|hostname` ...

**Context:** server config, virtual host

**Status:** Base

**Module:** mod_remoteip

The **RemoteIPTrustedProxy** directive adds one or more addresses (or address blocks) to trust as presenting a valid RemoteIPHeader value of the useragent IP. Unlike the **RemoteIPInternalProxy** directive, any intranet or private IP address reported by such proxies, including the 10/8, 172.16/12, 192.168/16, 169.254/16 and 127/8 blocks (or outside of the IPv6 public 2000::/3 block) are not trusted as the useragent IP, and are left in the **RemoteIPHeader** header's value.

**Trusted (Load Balancer) Example**

RemoteIPHeader X-Forwarded-For
RemoteIPTrustedProxy 10.0.2.16/28
RemoteIPTrustedProxy proxy.example.com
**Description:** Declare client intranet IP addresses trusted to present the RemoteIPHeader value

**Syntax:** RemoteIPTrustedProxyList *filename*

**Context:** server config, virtual host

**Status:** Base

**Module:** mod_remoteip

The **RemoteIPTrustedProxyList** directive specifies a file parsed at startup, and builds a list of addresses (or address blocks) to trust as presenting a valid RemoteIPHeader value of the useragent IP.

The '#' hash character designates a comment line, otherwise each whitespace or newline separated entry is processed identically to the **RemoteIPTrustedProxy** directive.

### Trusted (Load Balancer) Example
RemoteIPHeader X-Forwarded-For RemoteIPTrustedProxyList conf/trusted-proxies.lst

### conf/trusted-proxies.lst contents

```
# Identified external proxies;
192.0.2.16/28 #wap phone group of proxies
proxy.isp.example.com #some well known ISP
```
| **Description:** | Set timeout and minimum data rate for receiving requests |
| **Status:**     | Extension                                                   |
| **Module Identifier:** | reqtimeout_module                                     |
| **Source File:**  | mod_reqtimeout.c                                      |
| **Compatibility:** | Available in Apache HTTPD 2.2.15 and later                |
1. Allow 10 seconds to receive the request including the headers and 30 seconds for receiving the request body:

```
RequestReadTimeout header=10 body=30
```

2. Allow at least 10 seconds to receive the request body. If the client sends data, increase the timeout by 1 second for every 1000 bytes received, with no upper limit for the timeout (except for the limit given indirectly by `LimitRequestBody`):

```
RequestReadTimeout body=10,MinRate=1000
```

3. Allow at least 10 seconds to receive the request including the headers. If the client sends data, increase the timeout by 1 second for every 500 bytes received. But do not allow more than 30 seconds for the request including the headers:

```
RequestReadTimeout header=10-30,MinRate=500
```

4. Usually, a server should have both header and body timeouts configured. If a common configuration is used for http and https virtual hosts, the timeouts should not be set too low:

```
RequestReadTimeout header=20-40,MinRate=
```
**Description:** Set timeout values for receiving request headers and body from client.

**Syntax:** RequestReadTimeout [header=timeout[-maxtimeout]][,MinRate=rate] [body=timeout[-maxtimeout]][,MinRate=rate]

**Default:** header=20-40, MinRate=500
body=20, MinRate=500

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_reqtimeout

**Compatibility:** Available in version 2.2.15 and later; defaulted to disabled in version 2.3.14 and earlier.

This directive can set various timeouts for receiving the request headers and the request body from the client. If the client fails to send headers or body within the configured time, a 408 REQUEST TIME OUT error is sent.

For SSL virtual hosts, the header timeout values include the time needed to do the initial SSL handshake. If the user’s browser is configured to query certificate revocation lists and the CRL server is not reachable, the initial SSL handshake may take a significant time until the browser gives up waiting for the CRL. Therefore the header timeout values should not be set to very low values for SSL virtual hosts. The body timeout values include the time needed for SSL renegotiation (if necessary).

When an **AcceptFilter** is in use (usually the case on Linux and FreeBSD), the socket is not sent to the server process before at least one byte (or the whole request for httpready) is received. The header timeout configured with RequestReadTimeout is
only effective after the server process has received the socket.

For each of the two timeout types (header or body), there are three ways to specify the timeout:

- **Fixed timeout value:**

  ```
type=timeout
  
  The time in seconds allowed for reading all of the request headers or body, respectively. A value of 0 means no limit.
  ```

- **Disable module for a vhost:**

  ```
  header=0  body=0
  
  This disables mod_reqtimeout completely.
  ```

- **Timeout value that is increased when data is received:**

  ```
type=timeout,MinRate=data_rate
  
  Same as above, but whenever data is received, the timeout value is increased according to the specified minimum data rate (in bytes per second).
  ```

- **Timeout value that is increased when data is received, with an upper bound:**

  ```
type=timeout-maxtimeout,MinRate=data_rate
  
  Same as above, but the timeout will not be increased above the second value of the specified timeout range.
## Apache Module mod_request

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Filters to handle and make available HTTP request bodies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>request_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_request.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.3 and later</td>
</tr>
</tbody>
</table>
**Description:** Keep the request body instead of discarding it up to the specified maximum size, for potential use by filters such as mod_include.

**Syntax:** KeptBodySize maximum size in bytes

**Default:** KeptBodySize 0

**Context:** directory

**Status:** Base

**Module:** mod_request

Under normal circumstances, request handlers such as the default handler for static files will discard the request body when it is not needed by the request handler. As a result, filters such as mod_include are limited to making GET requests only when including other URLs as subrequests, even if the original request was a POST request, as the discarded request body is no longer available once filter processing is taking place.

When this directive has a value greater than zero, request handlers that would otherwise discard request bodies will instead set the request body aside for use by filters up to the maximum size specified. In the case of the mod_include filter, an attempt to POST a request to the static shtml file will cause any subrequests to be POST requests, instead of GET requests as before.

This feature makes it possible to break up complex web pages and web applications into small individual components, and combine the components and the surrounding web page structure together using mod_include. The components can take the form of CGI programs, scripted languages, or URLs reverse proxied into the URL space from another server using mod_proxy.

**Note:** Each request set aside has to be set aside in temporary RAM until the request is complete. As a result, care should be
taken to ensure sufficient RAM is available on the server to support the intended load. Use of this directive should be limited to where needed on targeted parts of your URL space, and with the lowest possible value that is still big enough to hold a request body.

If the request size sent by the client exceeds the maximum size allocated by this directive, the server will return 413 Request Entity Too Large.

**See also**

- [mod_include](http://httpd.apache.org/docs/2.4/mod/mod_include.html) documentation
- [mod_auth_form](http://httpd.apache.org/docs/2.4/mod/mod_auth_form.html) documentation
Apache Module mod_rewrite

| Description: | Provides a rule-based rewriting engine to rewrite requested URLs on the fly |
| Status: | Extension |
| Module Identifier: | rewrite_module |
| Source File: | mod_rewrite.c |

Summary

The mod_rewrite module uses a rule-based rewriting engine, based on a PCRE regular-expression parser, to rewrite requested URLs on the fly. By default, mod_rewrite maps a URL to a filesystem path. However, it can also be used to redirect one URL to another URL, or to invoke an internal proxy fetch.

mod_rewrite provides a flexible and powerful way to manipulate URLs using an unlimited number of rules. Each rule can have an unlimited number of attached rule conditions, to allow you to rewrite URL based on server variables, environment variables, HTTP headers, or time stamps.

mod_rewrite operates on the full URL path, including the path-info section. A rewrite rule can be invoked in httpd.conf or in .htaccess. The path generated by a rewrite rule can include a query string, or can lead to internal sub-processing, external request redirection, or internal proxy throughput.

Further details, discussion, and examples, are provided in the detailed mod_rewrite documentation.
**mod_rewrite** offers detailed logging of its actions at the trace1 to trace8 log levels. The log level can be set specifically for **mod_rewrite** using the **LogLevel** directive: Up to level debug, no actions are logged, while trace8 means that practically all actions are logged.

Using a high trace log level for **mod_rewrite** will slow down your Apache HTTP Server dramatically! Use a log level higher than trace2 only for debugging!

### Example

```
LogLevel alert rewrite:trace3
```

### RewriteLog

Those familiar with earlier versions of **mod_rewrite** will no doubt be looking for the **RewriteLog** and **RewriteLogLevel** directives. This functionality has been completely replaced by the new per-module logging configuration mentioned above.

To get just the **mod_rewrite**-specific log messages, pipe the log file through **grep**:

```
tail -f error_log|fgrep '[rewrite:'
```
The **RewriteBase** directive specifies the URL prefix to be used for per-directory (htaccess) **RewriteRule** directives that substitute a relative path.

This directive is *required* when you use a relative path in a substitution in per-directory (htaccess) context unless any of the following conditions are true:

- The original request, and the substitution, are underneath the **DocumentRoot** (as opposed to reachable by other means, such as **Alias**).
- The *filesystem* path to the directory containing the **RewriteRule**, suffixed by the relative substitution is also valid as a URL path on the server (this is rare).
- In Apache HTTP Server 2.4.16 and later, this directive may be omitted when the request is mapped via **Alias** or **mod_userdir**.

In the example below, **RewriteBase** is necessary to avoid rewriting to `http://example.com/opt/myapp-1.2.3/welcome.html` since the resource was not relative to the document root. This misconfiguration would normally cause the server to look for an "opt" directory under the document root.
DocumentRoot "/var/www/example.com"
AliasMatch "^/myapp" "/opt/myapp-1.2.3"
<Directory "/opt/myapp-1.2.3">
  RewriteEngine On
  RewriteBase "/myapp/"
  RewriteRule "^index\.html$" "welcome.html"
</Directory>
**Description:** Defines a condition under which rewriting will take place

**Syntax:** 
RewriteCond *TestString* *CondPattern* [*flags*]

**Context:** 
server config, virtual host, directory, .htaccess

**Override:** 
FileInfo

**Status:** 
Extension

**Module:** 
mod_rewrite

The **RewriteCond** directive defines a rule condition. One or more **RewriteCond** can precede a **RewriteRule** directive. The following rule is then only used if both the current state of the URI matches its pattern, **and** if these conditions are met.

*TestString* is a string which can contain the following expanded constructs in addition to plain text:

- **RewriteRule backreferences**: These are backreferences of the form $N (0 <= N <= 9). $1 to $9 provide access to the grouped parts (in parentheses) of the pattern, from the RewriteRule which is subject to the current set of RewriteCond conditions. $0 provides access to the whole string matched by that pattern.

- **RewriteCond backreferences**: These are backreferences of the form %N (0 <= N <= 9). %1 to %9 provide access to the grouped parts (again, in parentheses) of the pattern, from the last matched RewriteCond in the current set of conditions. %0 provides access to the whole string matched by that pattern.

- **RewriteMap expansions**: These are expansions of the form ${mapname: key|default}. See the documentation for RewriteMap for more details.
- **Server-Variables**: These are variables of the form `%{NAME_OF_VARIABLE}` where `NAME_OF_VARIABLE` can be a string taken from the following list:

<table>
<thead>
<tr>
<th>HTTP headers:</th>
<th>connection &amp; request:</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_ACCEPT</td>
<td>AUTH_TYPE</td>
</tr>
<tr>
<td>HTTP_COOKIE</td>
<td>CONN_REMOTE_ADDR</td>
</tr>
<tr>
<td>HTTP_FORWARDED</td>
<td>CONTEXT_PREFIX</td>
</tr>
<tr>
<td>HTTP_HOST</td>
<td>CONTEXT_DOCUMENT_RC</td>
</tr>
<tr>
<td>HTTP_PROXY_CONNECTION</td>
<td>IPV6</td>
</tr>
<tr>
<td>HTTP_REFERER</td>
<td>PATH_INFO</td>
</tr>
<tr>
<td>HTTP_USER_AGENT</td>
<td>QUERY_STRING</td>
</tr>
<tr>
<td></td>
<td>REMOTE_ADDR</td>
</tr>
<tr>
<td></td>
<td>REMOTE_HOST</td>
</tr>
<tr>
<td></td>
<td>REMOTE_IDENT</td>
</tr>
<tr>
<td></td>
<td>REMOTE_PORT</td>
</tr>
<tr>
<td></td>
<td>REMOTE_USER</td>
</tr>
<tr>
<td></td>
<td>REQUEST_METHOD</td>
</tr>
<tr>
<td></td>
<td>SCRIPT_FILENAME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>server internals:</th>
<th>date and time:</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOCUMENT_ROOT</td>
<td>TIME_YEAR</td>
</tr>
<tr>
<td>SCRIPT_GROUP</td>
<td>TIME_MON</td>
</tr>
<tr>
<td>SCRIPT_USER</td>
<td>TIME_DAY</td>
</tr>
<tr>
<td>SERVER_ADDR</td>
<td>TIME_HOUR</td>
</tr>
<tr>
<td>SERVER_ADMIN</td>
<td>TIME_MIN</td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td>TIME_SEC</td>
</tr>
<tr>
<td>SERVER_PORT</td>
<td>TIME_WDAY</td>
</tr>
<tr>
<td>SERVER_PROTOCOL</td>
<td>TIME</td>
</tr>
<tr>
<td>SERVER_SOFTWARE</td>
<td></td>
</tr>
</tbody>
</table>

These variables all correspond to the similarly named HTTP MIME-headers, C variables of the Apache HTTP Server or `struct tm` fields of the Unix system. Most are documented [here](#) or elsewhere in the Manual or in the CGI specification.
SERVER_NAME and SERVER_PORT depend on the values of UseCanonicalName and UseCanonicalPhysicalPort respectively.

Those that are special to mod_rewrite include those below.

**API_VERSION**
This is the version of the Apache httpd module API (the internal interface between server and module) in the current httpd build, as defined in include/ap_mmn.h. The module API version corresponds to the version of Apache httpd in use (in the release version of Apache httpd 1.3.14, for instance, it is 19990320:10), but is mainly of interest to module authors.

**CONN_REMOTE_ADDR**
Since 2.4.8: The peer IP address of the connection (see the mod_remoteip module).

**HTTPS**
Will contain the text "on" if the connection is using SSL/TLS, or "off" otherwise. (This variable can be safely used regardless of whether or not mod_ssl is loaded).

**IS_SUBREQ**
Will contain the text "true" if the request currently being processed is a sub-request, "false" otherwise. Sub-requests may be generated by modules that need to resolve additional files or URIs in order to complete their tasks.

**REMOTE_ADDR**
The IP address of the remote host (see the mod_remoteip module).

**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. Depending on the value of AcceptPathInfo, the server may have only used some leading components of the REQUEST_URI to map the request to a file.

REQUEST_SCHEME
Will contain the scheme of the request (usually "http" or "https"). This value can be influenced with ServerName.

REQUEST_URI
The path component of the requested URI, such as "/index.html". This notably excludes the query string which is available as its own variable named QUERY_STRING.

THE_REQUEST
The full HTTP request line sent by the browser to the server (e.g., "GET /index.html HTTP/1.1"). This does not include any additional headers sent by the browser. This value has not been unescaped (decoded), unlike most other variables below.

If the TestString has the special value expr, the CondPattern will be treated as an ap_expr. HTTP headers referenced in the expression will be added to the Vary header if the novary flag is not given.

Other things you should be aware of:

1. The variables SCRIPT_FILENAME and REQUEST_FILENAME contain the same value - the value of the filename field of the internal request_rec structure of the Apache HTTP Server. The first name is the commonly
known CGI variable name while the second is the appropriate counterpart of REQUEST_URI (which contains the value of the uri field of request_rec).

If a substitution occurred and the rewriting continues, the value of both variables will be updated accordingly.

If used in per-server context (i.e., before the request is mapped to the filesystem) SCRIPT_FILENAME and REQUEST_FILENAME cannot contain the full local filesystem path since the path is unknown at this stage of processing. Both variables will initially contain the value of REQUEST_URI in that case. In order to obtain the full local filesystem path of the request in per-server context, use an URL-based look-ahead %{LA-U:REQUEST_FILENAME} to determine the final value of REQUEST_FILENAME.

2. %{ENV:variable}, where variable can be any environment variable, is also available. This is looked-up via internal Apache httpd structures and (if not found there) via getenv() from the Apache httpd server process.

3. %{SSL:variable}, where variable is the name of an SSL environment variable, can be used whether or not mod_ssl is loaded, but will always expand to the empty string if it is not. Example: %{SSL:SSL_CIPHER_USEKEYSIZE} may expand to 128. These variables are available even without setting the StdEnvVars option of the SSLOptions directive.

4. %{HTTP:header}, where header can be any HTTP MIME-header name, can always be used to obtain the value of a header sent in the HTTP request. Example: %{HTTP:Proxy-Connection} is the value of the HTTP header ``Proxy-Connection:``. If a HTTP header is used in a condition this header is added
to the Vary header of the response in case the condition evaluates to true for the request. It is **not** added if the condition evaluates to false for the request. Adding the HTTP header to the Vary header of the response is needed for proper caching.

It has to be kept in mind that conditions follow a short circuit logic in the case of the '**ornext|OR**' flag so that certain conditions might not be evaluated at all.

5. `{LA-U:variable}` can be used for look-aheads which perform an internal (URL-based) sub-request to determine the final value of `variable`. This can be used to access variable for rewriting which is not available at the current stage, but will be set in a later phase.

   For instance, to rewrite according to the REMOTE_USER variable from within the per-server context (httpd.conf file) you must use `{LA-U:REMOTE_USER}` - this variable is set by the authorization phases, which come *after* the URL translation phase (during which mod_rewrite operates).

   On the other hand, because mod_rewrite implements its per-directory context (.htaccess file) via the Fixup phase of the API and because the authorization phases come *before* this phase, you just can use `{REMOTE_USER}` in that context.

6. `{LA-F:variable}` can be used to perform an internal (filename-based) sub-request, to determine the final value of `variable`. Most of the time, this is the same as LA-U above.

*CondPattern* is the condition pattern, a regular expression which is applied to the current instance of the `TestString`. `TestString` is first evaluated, before being matched against *CondPattern*.

*CondPattern* is usually a *perl compatible regular expression*, but
there is additional syntax available to perform other useful tests against the Teststring:

1. You can prefix the pattern string with a '! ' character (exclamation mark) to negate the result of the condition, no matter what kind of CondPattern is used.

2. You can perform lexicographical string comparisons:

   <CondPattern
   Lexicographically precedes
   Treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString lexicographically precedes CondPattern.

   >CondPattern
   Lexicographically follows
   Treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString lexicographically follows CondPattern.

   =CondPattern
   Lexicographically equal
   Treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString is lexicographically equal to CondPattern (the two strings are exactly equal, character for character). If CondPattern is "" (two quotation marks) this compares TestString to the empty string.

   <=CondPattern
   Lexicographically less than or equal to
   Treats the CondPattern as a plain string and compares it lexicographically to TestString. True if TestString lexicographically precedes CondPattern, or is equal to CondPattern (the two strings are equal, character for character).
>=CondPattern
Lexicographically greater than or equal to
Treats the CondPattern as a plain string and compares it
lexicographically to TestString. True if TestString
lexicographically follows CondPattern, or is equal to
CondPattern (the two strings are equal, character for
color).

3. You can perform integer comparisons:
   - eq
     Is numerically equal to
     The TestString is treated as an integer, and is numerically
     compared to the CondPattern. True if the two are
     numerically equal.
   - ge
     Is numerically greater than or equal to
     The TestString is treated as an integer, and is numerically
     compared to the CondPattern. True if the TestString
     is numerically greater than or equal to the CondPattern.
   - gt
     Is numerically greater than
     The TestString is treated as an integer, and is numerically
     compared to the CondPattern. True if the TestString
     is numerically greater than the CondPattern.
   - le
     Is numerically less than or equal to
     The TestString is treated as an integer, and is numerically
     compared to the CondPattern. True if the TestString
     is numerically less than or equal to the CondPattern. Avoid
     confusion with the -l by using the -L or -h variant.
   - lt
     Is numerically less than
     The TestString is treated as an integer, and is numerically
compared to the CondPattern. True if the TestString is numerically less than the CondPattern. Avoid confusion with the -l by using the -L or -h variant.

-ne
Is numerically not equal to
The TestString is treated as an integer, and is numerically compared to the CondPattern. True if the two are numerically different. This is equivalent to ! -eq.

4. You can perform various file attribute tests:

-d
Is directory.
Treats the TestString as a pathname and tests whether or not it exists, and is a directory.

-f
Is regular file.
Treats the TestString as a pathname and tests whether or not it exists, and is a regular file.

-F
Is existing file, via subrequest.
Checks whether or not TestString 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!

-h
Is symbolic link, bash convention.
See -l.

-l
Is symbolic link.
Treats the TestString as a pathname and tests whether or not it exists, and is a symbolic link. May also use the
bash convention of -L or -h if there's a possibility of confusion such as when using the -lt or -le tests.

-L
Is symbolic link, bash convention.
See -l.

-s
Is regular file, with size.
Treats the TestString as a pathname and tests whether or not it exists, and is a regular file with size greater than zero.

-U
Is existing URL, via subrequest.
Checks whether or not TestString 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!

This flag only returns information about things like access control, authentication, and authorization. This flag does not return information about the status code the configured handler (static file, CGI, proxy, etc.) would have returned.

-x
Has executable permissions.
Treats the TestString as a pathname and tests whether or not it exists, and has executable permissions. These permissions are determined according to the underlying OS.

For example:

RewriteCond /var/www/%{REQUEST_URI} !-f
5. If the *TestString* has the special value *expr*, the *CondPattern* will be treated as an *ap_expr*.

In the below example, -strmatch is used to compare the REFERER against the site hostname, to block unwanted hotlinking.

```plaintext
RewriteRule ^(.+) /other/archive/$1 [R]

RewriteCond expr "! %{HTTP_REFERER} -strmatch '*://%{HTTP_HOST}/*'"
RewriteRule "^/images" "-" [F]
```

You can also set special flags for *CondPattern* by appending [flags] as the third argument to the RewriteCond directive, where flags is a comma-separated list of any of the following flags:

- **'nocase|NC' (no case)**
  This makes the test case-insensitive - differences between 'A-Z' and 'a-z' are ignored, both in the expanded *TestString* and the *CondPattern*. This flag is effective only for comparisons between *TestString* and *CondPattern*. It has no effect on filesystem and subrequest checks.
- **'ornext|OR' (or next condition)**
  Use this to combine rule conditions with a local OR instead of the implicit AND. Typical example:

```plaintext
RewriteCond %{REMOTE_HOST} "^host1"
RewriteCond %{REMOTE_HOST} "^host2"
RewriteCond %{REMOTE_HOST} "^host3"
RewriteRule ...some special stuff for an
```

Without this flag you would have to write the condition/rule
pair three times.

- **'novary|NV' (no vary)**
  If a HTTP header is used in the condition, this flag prevents this header from being added to the Vary header of the response.
  Using this flag might break proper caching of the response if the representation of this response varies on the value of this header. So this flag should be only used if the meaning of the Vary header is well understood.

**Example:**

To rewrite the Homepage of a site according to the ``User-Agent:`` header of the request, you can use the following:

```apache
RewriteCond %{HTTP_USER_AGENT} "(iPhone|Blackberry|Android)"
RewriteRule ^/$ /homepage.mobile.html [L]
RewriteRule ^/$ /homepage.std.html [L]
```

**Explanation:** If you use a browser which identifies itself as a mobile browser (note that the example is incomplete, as there are many other mobile platforms), the mobile version of the homepage is served. Otherwise, the standard page is served.
RewriteEngine directive enables or disables the runtime rewriting engine. If it is set to off this module does no runtime processing at all. It does not even update the SCRIPT_URx environment variables.

Use this directive to disable rules in a particular context, rather than commenting out all the RewriteRule directives.

Note that rewrite configurations are not inherited by virtual hosts. This means that you need to have a RewriteEngine on directive for each virtual host in which you wish to use rewrite rules.

RewriteMap directives of the type prg are not started during server initialization if they're defined in a context that does not have RewriteEngine set to on
The `RewriteMap` directive defines a **Rewriting Map** which can be used inside rule substitution strings by the mapping-functions to insert/substitute fields through a key lookup. The source of this lookup can be of various types.

The `MapName` is the name of the map and will be used to specify a mapping-function for the substitution strings of a rewriting rule via one of the following constructs:

\[
\begin{align*}
\&\{ \text{MapName} : \text{LookupKey} \} \\
\&\{ \text{MapName} : \text{LookupKey} | \text{DefaultValue} \}
\end{align*}
\]

When such a construct occurs, the map `MapName` is consulted and the key `LookupKey` is looked-up. If the key is found, the mapping-function construct is substituted by `SubstValue`. If the key is not found then it is substituted by `DefaultValue` or by the empty string if no `DefaultValue` was specified. Empty values behave as if the key was absent, therefore it is not possible to distinguish between empty-valued keys and absent keys.

For example, you might define a `RewriteMap` as:

```
RewriteMap examplemap "txt:/path/to/file/map.txt"
```

You would then be able to use this map in a `RewriteRule` as follows:
The following combinations for \textit{MapType} and \textit{MapSource} can be used:

\textbf{txt}

A plain text file containing space-separated key-value pairs, one per line. (Details ...)

\textbf{rnd}

Randomly selects an entry from a plain text file (Details ...)

\textbf{dbm}

Looks up an entry in a dbm file containing name, value pairs. Hash is constructed from a plain text file format using the \texttt{htxt2dbm} utility. (Details ...)

\textbf{int}

One of the four available internal functions provided by \texttt{RewriteMap}: toupper, tolower, escape or unescape. (Details ...)

\textbf{prg}

Calls an external program or script to process the rewriting. (Details ...)

\textbf{dbd} or \textbf{fastdbd}

A SQL SELECT statement to be performed to look up the rewrite target. (Details ...)

Further details, and numerous examples, may be found in the \texttt{RewriteMap HowTo}
The **RewriteOptions** directive sets some special options for the current per-server or per-directory configuration. The *Option* string can currently only be one of the following:

**Inherit**

This forces the current configuration to inherit the configuration of the parent. In per-virtual-server context, this means that the maps, conditions and rules of the main server are inherited. In per-directory context this means that conditions and rules of the parent directory's `.htaccess` configuration or `<Directory>` sections are inherited. The inherited rules are virtually copied to the section where this directive is being used. If used in combination with local rules, the inherited rules are copied behind the local rules. The position of this directive - below or above of local rules - has no influence on this behavior. If local rules forced the rewriting to stop, the inherited rules won't be processed.

Rules inherited from the parent scope are applied **after** rules specified in the child scope.

**InheritBefore**

Like Inherit above, but the rules from the parent scope are applied **before** rules specified in the child scope. Available in Apache HTTP Server 2.3.10 and later.
**InheritDown**
If this option is enabled, all child configurations will inherit the configuration of the current configuration. It is equivalent to specifying RewriteOptions Inherit in all child configurations. See the Inherit option for more details on how the parent-child relationships are handled. Available in Apache HTTP Server 2.4.8 and later.

**InheritDownBefore**
Like InheritDown above, but the rules from the current scope are applied before rules specified in any child's scope. Available in Apache HTTP Server 2.4.8 and later.

**IgnoreInherit**
This option forces the current and child configurations to ignore all rules that would be inherited from a parent specifying InheritDown or InheritDownBefore. Available in Apache HTTP Server 2.4.8 and later.

**AllowNoSlash**
By default, mod_rewrite will ignore URLs that map to a directory on disk but lack a trailing slash, in the expectation that the mod_dir module will issue the client with a redirect to the canonical URL with a trailing slash.

When the DirectorySlash directive is set to off, the AllowNoSlash option can be enabled to ensure that rewrite rules are no longer ignored. This option makes it possible to apply rewrite rules within .htaccess files that match the directory without a trailing slash, if so desired. Available in Apache HTTP Server 2.4.0 and later.

**AllowAnyURI**
When RewriteRule is used in VirtualHost or server context with version 2.2.22 or later of httpd, mod_rewrite
will only process the rewrite rules if the request URI is a URL-path. This avoids some security issues where particular rules could allow "surprising" pattern expansions (see CVE-2011-3368 and CVE-2011-4317). To lift the restriction on matching a URL-path, the AllowAnyURI option can be enabled, and `mod_rewrite` will apply the rule set to any request URI string, regardless of whether that string matches the URL-path grammar required by the HTTP specification. Available in Apache HTTP Server 2.4.3 and later.

**Security Warning**

Enabling this option will make the server vulnerable to security issues if used with rewrite rules which are not carefully authored. It is strongly recommended that this option is not used. In particular, beware of input strings containing the '@' character which could change the interpretation of the transformed URI, as per the above CVE names.

**MergeBase**

With this option, the value of `RewriteBase` is copied from where it's explicitly defined into any sub-directory or sub-location that doesn't define its own `RewriteBase`. This was the default behavior in 2.4.0 through 2.4.3, and the flag to restore it is available Apache HTTP Server 2.4.4 and later.

**IgnoreContextInfo**

When a relative substitution is made in directory (htaccess) context and `RewriteBase` has not been set, this module uses some extended URL and filesystem context information to change the relative substitution back into a URL. Modules such as `mod_userdir` and `mod_alias` supply this extended context info. Available in 2.4.16 and later.
**LegacyPrefixDocRoot**

Prior to 2.4.26, if a substitution was an absolute URL that matched the current virtual host, the URL might first be reduced to a URL-path and then later reduced to a local path. Since the URL can be reduced to a local path, the path should be prefixed with the document root. This prevents a file such as `/tmp/myfile` from being accessed when a request is made to `http://host/file/myfile` with the following `RewriteRule`.

```
RewriteRule /file/(.*) http://localhost/
```

This option allows the old behavior to be used where the document root is not prefixed to a local path that was reduced from a URL. Available in 2.4.26 and later.
The **RewriteRule** directive is the real rewriting workhorse. The directive can occur more than once, with each instance defining a single rewrite rule. The order in which these rules are defined is important - this is the order in which they will be applied at runtime.

*Pattern* is a perl compatible regular expression. What this pattern is compared against varies depending on where the **RewriteRule** directive is defined.

**What is matched?**

- In **VirtualHost** context, the *Pattern* will initially be matched against the part of the URL after the hostname and port, and before the query string (e.g. "/app1/index.html"). This is the (%-decoded) **URL-path**.

- In per-directory context (**Directory** and .htaccess), the *Pattern* is matched against only a partial path, for example a request of "/app1/index.html" may result in comparison against "app1/index.html" or "index.html" depending on where the **RewriteRule** is defined.

The directory path where the rule is defined is stripped from the currently mapped filesystem path before comparison (up to and including a trailing slash). The net result of this...
per-directory prefix stripping is that rules in this context only match against the portion of the currently mapped filesystem path "below" where the rule is defined.

Directives such as DocumentRoot and Alias, or even the result of previous RewriteRule substitutions, determine the currently mapped filesystem path.

- If you wish to match against the hostname, port, or query string, use a RewriteCond with the %{HTTP_HOST}, %{SERVER_PORT}, or %{QUERY_STRING} variables respectively.

### Per-directory Rewrites

- The rewrite engine may be used in .htaccess files and in <Directory> sections, with some additional complexity.
- To enable the rewrite engine in this context, you need to set "RewriteEngine On" and "Options FollowSymLinks" must be enabled. If your administrator has disabled override of FollowSymLinks for a user's directory, then you cannot use the rewrite engine. This restriction is required for security reasons.
- See the RewriteBase directive for more information regarding what prefix will be added back to relative substitutions.
- If you wish to match against the full URL-path in a per-directory (htaccess) RewriteRule, use the %{REQUEST_URI} variable in a RewriteCond.
- The removed prefix always ends with a slash, meaning the matching occurs against a string which never has a leading slash. Therefore, a Pattern with ^/ never matches in per-directory context.
- Although rewrite rules are syntactically permitted in
<Location> and <Files> sections (including their regular expression counterparts), this should never be necessary and is unsupported. A likely feature to break in these contexts is relative substitutions.

For some hints on regular expressions, see the mod_rewrite Introduction.

In mod_rewrite, the NOT character (`!'`) is also available as a possible pattern prefix. This enables you to negate a pattern; to say, for instance: ``if the current URL does NOT match this pattern''. This can be used for exceptional cases, where it is easier to match the negative pattern, or as a last default rule.

**Note**

When using the NOT character to negate a pattern, you cannot include grouped wildcard parts in that pattern. This is because, when the pattern does NOT match (ie, the negation matches), there are no contents for the groups. Thus, if negated patterns are used, you cannot use $N in the substitution string!

The Substitution of a rewrite rule is the string that replaces the original URL-path that was matched by Pattern. The Substitution may be a:

**file-system path**

Designates the location on the file-system of the resource to be delivered to the client. Substitutions are only treated as a file-system path when the rule is configured in server (virtualhost) context and the first component of the path in the substitution exists in the file-system

**URL-path**

A DocumentRoot-relative path to the resource to be served.
Note that **mod_rewrite** tries to guess whether you have specified a file-system path or a URL-path by checking to see if the first segment of the path exists at the root of the file-system. For example, if you specify a *Substitution* string of `/www/file.html`, then this will be treated as a URL-path *unless* a directory named `www` exists at the root or your file-system (or, in the case of using rewrites in a `.htaccess` file, relative to your document root), in which case it will be treated as a file-system path. If you wish other URL-mapping directives (such as `Alias`) to be applied to the resulting URL-path, use the [PT] flag as described below.

**Absolute URL**

If an absolute URL is specified, **mod_rewrite** checks to see whether the hostname matches the current host. If it does, the scheme and hostname are stripped out and the resulting path is treated as a URL-path. Otherwise, an external redirect is performed for the given URL. To force an external redirect back to the current host, see the [R] flag below.

- **(dash)**
  
  A dash indicates that no substitution should be performed (the existing path is passed through untouched). This is used when a flag (see below) needs to be applied without changing the path.

In addition to plain text, the *Substitution* string can include

1. back-references ($N) to the RewriteRule pattern
2. back-references (%N) to the last matched RewriteCond pattern
3. server-variables as in rule condition test-strings (% {VARNAME})
4. [mapping-function](calls).${mapname:|default}
Back-references are identifiers of the form $\mathbf{N}$ ($\mathbf{N}=0..9$), which will be replaced by the contents of the $\mathbf{N}$th group of the matched Pattern. The server-variables are the same as for the TestString of a RewriteCond directive. The mapping-functions come from the RewriteMap directive and are explained there. These three types of variables are expanded in the order above.

Rewrite rules are applied to the results of previous rewrite rules, in the order in which they are defined in the config file. The URL-path or file-system path (see "What is matched?", above) is completely replaced by the Substitution and the rewriting process continues until all rules have been applied, or it is explicitly terminated by an L_flag, or other flag which implies immediate termination, such as END or F.

### Modifying the Query String

By default, the query string is passed through unchanged. You can, however, create URLs in the substitution string containing a query string part. Simply use a question mark inside the substitution string to indicate that the following text should be re-injected into the query string. When you want to erase an existing query string, end the substitution string with just a question mark. To combine new and old query strings, use the [QSA] flag.

Additionally you can set special actions to be performed by appending [flags] as the third argument to the RewriteRule directive. Flags is a comma-separated list, surround by square brackets, of any of the flags in the following table. More details, and examples, for each flag, are available in the Rewrite Flags document.

<table>
<thead>
<tr>
<th>Flag and syntax</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>Escape non-alphanumeric characters in backreferences before applying the transformation. [details]</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>backrefnoplus</td>
<td>If backreferences are being escaped, spaces are escaped to %20 instead of +. Useful when a backreference will be used in the path component rather than the query string. [details]</td>
</tr>
<tr>
<td>chain</td>
<td>Rule is chained to the following rule. If the rule(s) chained to it will be skipped. [details]</td>
</tr>
<tr>
<td>cookie</td>
<td>Sets a cookie in the client browser. Full syntax: CO=NAME:VAL:domain[:lifetime]:path[:secure] [details]</td>
</tr>
<tr>
<td>discardpath</td>
<td>Causes the PATH_INFO portion of the rewritten URI to be discarded. [details]</td>
</tr>
<tr>
<td>END</td>
<td>Stop the rewriting process immediately and don't apply any more rules. Also prevents further execution of rules in per-directory and .htaccess context (available in 2.3.9 and later) [details]</td>
</tr>
<tr>
<td>env</td>
<td>Causes an environment variable VAR to be set (to the value VAL if provided). The form !VAR causes the environment variable VAR to be unset. [details]</td>
</tr>
<tr>
<td>forbidden</td>
<td>Returns a 403 FORBIDDEN response to the client browser. [details]</td>
</tr>
<tr>
<td>gone</td>
<td>Returns a 410 GONE response to the client browser. [details]</td>
</tr>
<tr>
<td>Handler</td>
<td>Causes the resulting URI to be sent to the Content-handler for processing. [details]</td>
</tr>
<tr>
<td>last</td>
<td>Stop the rewriting process immediately and don't apply any more rules. Especially note caveats for per-directory and .htaccess context (see also the END flag).</td>
</tr>
<tr>
<td>next</td>
<td>Re-run the rewriting process, starting again with the first rule, using the result of the ruleset so far as a starting point. [details]</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td>nocase</td>
<td>Makes the pattern comparison case-insensitive.</td>
</tr>
<tr>
<td>noescape</td>
<td>Prevent <code>mod_rewrite</code> from applying hexcode escaping to special characters in the result of the rewrite.</td>
</tr>
<tr>
<td>nosubreq</td>
<td>Causes a rule to be skipped if the current request is an internal sub-request.</td>
</tr>
<tr>
<td>proxy</td>
<td>Force the substitution URL to be internally a proxy request.</td>
</tr>
<tr>
<td>passthrough</td>
<td>Forces the resulting URI to be passed back to the URL mapping engine for processing of other URI translators, such as <code>Alias</code> or <code>Redirect</code>.</td>
</tr>
<tr>
<td>qsappend</td>
<td>Appends any query string from the original request to any query string created in the rewrite target.</td>
</tr>
<tr>
<td>qsdiscard</td>
<td>Discard any query string attached to the incoming URI.</td>
</tr>
<tr>
<td>qslast</td>
<td>Interpret the last (right-most) question mark as the query string delimiter, instead of the first (left-most) as normally used. Available in 2.4.19 and later.</td>
</tr>
<tr>
<td>redirect</td>
<td>Forces an external redirect, optionally with an HTTP status code.</td>
</tr>
<tr>
<td>skip</td>
<td>Tells the rewriting engine to skip the next rule if the current rule matches.</td>
</tr>
<tr>
<td>type</td>
<td>Force the MIME-type of the target file to be a specific type.</td>
</tr>
</tbody>
</table>

**Home directory expansion**

When the substitution string begins with a string resembling "/~user" (via explicit text or backreferences), `mod_rewrite` performs home directory expansion independent of the presence or configuration of `mod_userdir`.

This expansion does not occur when the PT flag is used on the `RewriteRule` directive.
Here are all possible substitution combinations and their meanings:

**Inside per-server configuration (`httpd.conf`) for request ```GET /somepath/pathinfo```:**

<table>
<thead>
<tr>
<th>Given Rule</th>
<th>Resulting Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>^/somepath(.* ) otherpath$1</code></td>
<td>invalid, not supported</td>
</tr>
<tr>
<td><code>^/somepath(.* ) otherpath$1 [R]</code></td>
<td>invalid, not supported</td>
</tr>
<tr>
<td><code>^/somepath(.* ) otherpath$1 [P]</code></td>
<td>invalid, not supported</td>
</tr>
<tr>
<td><code>^/somepath(.* ) otherpath$1</code></td>
<td>/otherpath/pathinfo</td>
</tr>
<tr>
<td><code>^/somepath(.* ) otherpath$1 [P]</code></td>
<td>doesn't make sense, not supported</td>
</tr>
<tr>
<td><code>^/somepath(.* ) http://thishost/otherpath$1</code></td>
<td>/otherpath/pathinfo</td>
</tr>
<tr>
<td><code>^/somepath(.* ) http://thishost/otherpath$1 [P]</code></td>
<td>doesn't make sense, not supported</td>
</tr>
<tr>
<td><code>^/somepath(.* ) http://otherhost/otherpath$1</code></td>
<td><a href="http://otherhost/otherpath/pathinfo">http://otherhost/otherpath/pathinfo</a> via external redirection</td>
</tr>
<tr>
<td><code>^/somepath(.* ) </code></td>
<td><a href="http://otherhost/otherpath/pathinfo">http://otherhost/otherpath/pathinfo</a></td>
</tr>
</tbody>
</table>
Inside per-directory configuration for /somepath
(/physical/path/to/somepath/.htaccess, with
RewriteBase "/somepath")
for request `GET /somepath/localpath/pathinfo``:

<table>
<thead>
<tr>
<th>Given Rule</th>
<th>Resulting Substitution</th>
</tr>
</thead>
<tbody>
<tr>
<td>^localpath(.*) otherpath$1</td>
<td>/somepath/otherpath/pathinfo</td>
</tr>
<tr>
<td>^localpath(.*) otherpath$1</td>
<td><a href="http://thishost/somepath/otherpath/pathinfo">http://thishost/somepath/otherpath/pathinfo</a> via external redirection</td>
</tr>
<tr>
<td>^localpath(.*) otherpath$1</td>
<td>doesn't make sense, not supported</td>
</tr>
<tr>
<td>^localpath(.*) /otherpath$1</td>
<td>/otherpath/pathinfo</td>
</tr>
<tr>
<td>^localpath(.*) /otherpath$1</td>
<td><a href="http://thishost/otherpath/pathinfo">http://thishost/otherpath/pathinfo</a> via external redirection</td>
</tr>
<tr>
<td>^localpath(.*) /otherpath$1</td>
<td>doesn't make sense, not supported</td>
</tr>
<tr>
<td>^localpath(.) <a href="http://thishost/otherpath$1">http://thishost/otherpath$1</a></td>
<td>/otherpath/pathinfo</td>
</tr>
<tr>
<td>^localpath(.) <a href="http://thishost/otherpath$1">http://thishost/otherpath$1</a></td>
<td>doesn't make sense, not supported</td>
</tr>
<tr>
<td>^localpath(.) <a href="http://otherhost/otherpath$1">http://otherhost/otherpath$1</a></td>
<td><a href="http://otherhost/otherpath/pathinfo">http://otherhost/otherpath/pathinfo</a> via external redirection (the [R] flag is redundant)</td>
</tr>
</tbody>
</table>
| ^localpath(.*?)
| http://otherhost/otherpath$1
| [P] |
| http://otherhost/otherpath/pathinfo via internal proxy

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](https://www.apache.org/licenses/LICENSE-2.0).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)
Apache Module mod_sed

<table>
<thead>
<tr>
<th>Description:</th>
<th>Filter Input (request) and Output (response) content using sed syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Experimental</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>sed_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_sed.c sed0.c sed1.c regexp.c regexp.h sed.h</td>
</tr>
<tr>
<td>Compatibility:</td>
<td>Available in Apache 2.3 and later</td>
</tr>
</tbody>
</table>

Summary

mod_sed is an in-process content filter. The mod_sed filter implements the sed editing commands implemented by the Solaris 10 sed program as described in the manual page. However, unlike sed, mod_sed doesn't take data from standard input. Instead, the filter acts on the entity data sent between client and server. mod_sed can be used as an input or output filter. mod_sed is a content filter, which means that it cannot be used to modify client or server http headers.

The mod_sed output filter accepts a chunk of data, executes the sed scripts on the data, and generates the output which is passed to the next filter in the chain.

The mod_sed input filter reads the data from the next filter in the chain, executes the sed scripts, and returns the generated data to the caller filter in the filter chain.

Both the input and output filters only process the data if newline characters are seen in the content. At the end of the data, the rest of the data is treated as the last line.

A tutorial article on mod_sed, and why it is more powerful than simple string or regular expression search and replace, is available on the
author's blog.
Adding an output filter

# In the following example, the sed filter will change the string "monday" to "MON" and the string "sunday" to SUN in html documents before sending to the client.

<Directory "/var/www/docs/sed">
  AddOutputFilter Sed html
  OutputSed "s/monday/MON/g"
  OutputSed "s/sunday/SUN/g"
</Directory>

Adding an input filter

# In the following example, the sed filter will change the string "monday" to "MON" and the string "sunday" to SUN in the POST data sent to PHP.

<Directory "/var/www/docs/sed">
  AddInputFilter Sed php
  InputSed "s/monday/MON/g"
  InputSed "s/sunday/SUN/g"
</Directory>
Complete details of the sed command can be found from the [sed manual page](#).

**b**
Branch to the label specified (similar to goto).

**h**
Copy the current line to the hold buffer.

**H**
Append the current line to the hold buffer.

**g**
Copy the hold buffer to the current line.

**G**
Append the hold buffer to the current line.

**x**
Swap the contents of the hold buffer and the current line.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Sed command to filter request data (typically POST data)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>InputSed <em>sed-command</em></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>directory, .htaccess</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_sed</td>
</tr>
</tbody>
</table>

The **InputSed** directive specifies the *sed* command to execute on the request data e.g., POST data.
**Description:** Sed command for filtering response content

**Syntax:** OutputSed *sed-command*

**Context:** directory, .htaccess

**Status:** Experimental

**Module:** mod_sed

The **OutputSed** directive specifies the *sed* command to execute on the response.
Apache Module mod_session

**Description:** Session support

**Status:** Extension

**Module Identifier:** session_module

**Source File:** mod_session.c

**Compatibility:** Available in Apache 2.3 and later

### Summary

#### Warning

The session modules make use of HTTP cookies, and as such can fall victim to Cross Site Scripting attacks, or expose potentially private information to clients. Please ensure that the relevant risks have been taken into account before enabling the session functionality on your server.

This module provides support for a server wide per user session interface. Sessions can be used for keeping track of whether a user has been logged in, or for other per user information that should be kept available across requests.

Sessions may be stored on the server, or may be stored on the browser. Sessions may also be optionally encrypted for added security. These features are divided into several modules in addition to `mod_session`, `mod_session_crypto`, `mod_session_cookie` and `mod_session_dbd`. Depending on the server requirements, load the appropriate modules into the server (either statically at compile time or dynamically via the `LoadModule` directive).

Sessions may be manipulated from other modules that depend on the session, or the session may be read from and written to using
environment variables and HTTP headers, as appropriate.

**Bugfix checklist**

* httpd changelog
* Known issues
* Report a bug

**See also**

* mod_session_cookie
* mod_session_crypto
* mod_session_dbd
At the core of the session interface is a table of key and value pairs that are made accessible across browser requests. These pairs can be set to any valid string, as needed by the application making use of the session.

The "session" is a application/x-www-form-urlencoded string containing these key value pairs, as defined by the HTML specification.

The session can optionally be encrypted and base64 encoded before being written to the storage mechanism, as defined by the administrator.
The session interface is primarily developed for the use by other server modules, such as `mod_auth_form`, however CGI based applications can optionally be granted access to the contents of the session via the HTTP_SESSION environment variable. Sessions have the option to be modified and/or updated by inserting an HTTP response header containing the new session parameters.
Apache can be configured to keep track of per user sessions stored on a particular server or group of servers. This functionality is similar to the sessions available in typical application servers.

If configured, sessions are tracked through the use of a session ID that is stored inside a cookie, or extracted from the parameters embedded within the URL query string, as found in a typical GET request.

As the contents of the session are stored exclusively on the server, there is an expectation of privacy of the contents of the session. This does have performance and resource implications should a large number of sessions be present, or where a large number of webservers have to share sessions with one another.

The mod_session_dbd module allows the storage of user sessions within a SQL database via mod_dbd.
In high traffic environments where keeping track of a session on a server is too resource intensive or inconvenient, the option exists to store the contents of the session within a cookie on the client browser instead.

This has the advantage that minimal resources are required on the server to keep track of sessions, and multiple servers within a server farm have no need to share session information.

The contents of the session however are exposed to the client, with a corresponding risk of a loss of privacy. The mod_session_crypto module can be configured to encrypt the contents of the session before writing the session to the client.

The mod_session_cookie allows the storage of user sessions on the browser within an HTTP cookie.
Creating a session is as simple as turning the session on, and deciding where the session will be stored. In this example, the session will be stored on the browser, in a cookie called `session`.

### Browser based session

```plaintext
Session On
SessionCookieName session path=/
```

The session is not useful unless it can be written to or read from. The following example shows how values can be injected into the session through the use of a predetermined HTTP response header called `X-Replace-Session`.

### Writing to a session

```plaintext
Session On
SessionCookieName session path=/
SessionHeader X-Replace-Session
```

The header should contain name value pairs expressed in the same format as a query string in a URL, as in the example below. Setting a key to the empty string has the effect of removing that key from the session.

### CGI to write to a session

```bash
#!/bin/bash
echo "Content-Type: text/plain"
echo "X-Replace-Session: key1=foo&key2=&key3=bar"
echo
env
```

If configured, the session can be read back from the `HTTP_SESSION` environment variable. By default, the session is kept private, so this has to be explicitly turned on with the `SessionEnv` directive.
Read from a session

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session On</td>
<td>On</td>
</tr>
<tr>
<td>SessionEnv On</td>
<td>On</td>
</tr>
<tr>
<td>SessionCookieName session</td>
<td>path=/</td>
</tr>
<tr>
<td>SessionHeader</td>
<td>X-Replace-Session</td>
</tr>
</tbody>
</table>

Once read, the CGI variable HTTP_SESSION should contain the value `key1=foo&key3=bar.`
Using the "show cookies" feature of your browser, you would have seen a clear text representation of the session. This could potentially be a problem should the end user need to be kept unaware of the contents of the session, or where a third party could gain unauthorised access to the data within the session.

The contents of the session can be optionally encrypted before being placed on the browser using the `mod_session_crypto` module.

### Browser based encrypted session

- **Session On**
- **SessionCryptoPassphrase secret**
- **SessionCookieName session path=/**

The session will be automatically decrypted on load, and encrypted on save by Apache, the underlying application using the session need have no knowledge that encryption is taking place.

Sessions stored on the server rather than on the browser can also be encrypted as needed, offering privacy where potentially sensitive information is being shared between webservers in a server farm using the `mod_session_dbd` module.
The HTTP cookie mechanism also offers privacy features, such as the ability to restrict cookie transport to SSL protected pages only, or to prevent browser based javascript from gaining access to the contents of the cookie.

**Warning**

Some of the HTTP cookie privacy features are either non-standard, or are not implemented consistently across browsers. The session modules allow you to set cookie parameters, but it makes no guarantee that privacy will be respected by the browser. If security is a concern, use the `mod_session_crypto` to encrypt the contents of the session, or store the session on the server using the `mod_session_dbd` module.

Standard cookie parameters can be specified after the name of the cookie, as in the example below.

**Setting cookie parameters**

```
Session On
SessionCryptoPassphrase secret
SessionCookieName session path=/private;domain=example.com;httponly;secure;
```

In cases where the Apache server forms the frontend for backend origin servers, it is possible to have the session cookies removed from the incoming HTTP headers using the `SessionCookieRemove` directive. This keeps the contents of the session cookies from becoming accessible from the backend server.
As is possible within many application servers, authentication modules can use a session for storing the username and password after login. The \texttt{mod\ auth\ form} saves the user's login name and password within the session.

\begin{verbatim}
Form based authentication
Session On
SessionCryptoPassphrase secret
SessionCookieName session path=/
AuthFormProvider file
AuthUserFile "conf/passwd"
AuthType form
AuthName realm
#...
\end{verbatim}

See the \texttt{mod\ auth\ form} module for documentation and complete examples.
Integrating Sessions with External Applications

In order for sessions to be useful, it must be possible to share the contents of a session with external applications, and it must be possible for an external application to write a session of its own.

A typical example might be an application that changes a user's password set by `mod_auth_form`. This application would need to read the current username and password from the session, make the required changes to the user's password, and then write the new password to the session in order to provide a seamless transition to the new password.

A second example might involve an application that registers a new user for the first time. When registration is complete, the username and password is written to the session, providing a seamless transition to being logged in.

**Apache modules**

Modules within the server that need access to the session can use the `mod_session.h` API in order to read from and write to the session. This mechanism is used by modules like `mod_auth_form`.

**CGI programs and scripting languages**

Applications that run within the webserver can optionally retrieve the value of the session from the `HTTP_SESSION` environment variable. The session should be encoded as an `application/x-www-form-urlencoded` string as described by the [HTML specification](https://www.w3.org/html/wg/drafts/html/master/). The environment variable is controlled by the setting of the `SessionEnv` directive. The session can be written to by the script by returning a `application/x-www-form-urlencoded` response header with a name set by the `SessionHeader` directive. In both cases, any encryption or decryption, and the reading the session from or writing the session to the chosen storage mechanism...
is handled by the `mod_session` modules and corresponding configuration.

**Applications behind `mod_proxy`**

If the `SessionHeader` directive is used to define an HTTP request header, the session, encoded as an `application/x-www-form-urlencoded` string, will be made available to the application. If the same header is provided in the response, the value of this response header will be used to replace the session. As above, any encryption or decryption, and the reading the session from or writing the session to the chosen storage mechanism is handled by the `mod_session` modules and corresponding configuration.

**Standalone applications**

Applications might choose to manipulate the session outside the control of the Apache HTTP server. In this case, it is the responsibility of the application to read the session from the chosen storage mechanism, decrypt the session, update the session, encrypt the session and write the session to the chosen storage mechanism, as appropriate.
### Description
Enables a session for the current directory or location

### Syntax
Session On|Off

### Default
Session Off

### Context
server config, virtual host, directory, .htaccess

### Override
AuthConfig

### Status
Extension

### Module
mod_session

The **Session** directive enables a session for the directory or location container. Further directives control where the session will be stored and how privacy is maintained.
**Description:** Control whether the contents of the session are written to the `HTTP_SESSION` environment variable

**Syntax:** `SessionEnv On|Off`

**Default:** `SessionEnv Off`

**Context:** server config, virtual host, directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_session

If set to *On*, the `SessionEnv` directive causes the contents of the session to be written to a CGI environment variable called `HTTP_SESSION`.

The string is written in the URL query format, for example:

```
key1=foo&key3=bar
```
Description: Define URL prefixes for which a session is ignored
Syntax: SessionExclude path
Default: none
Context: server config, virtual host, directory, .htaccess
Status: Extension
Module: mod_session

The SessionExclude directive allows sessions to be disabled relative to URL prefixes only. This can be used to make a website more efficient, by targeting a more precise URL space for which a session should be maintained. By default, all URLs within the directory or location are included in the session. The SessionExclude directive takes precedence over the SessionInclude directive.

Warning
This directive has a similar purpose to the path attribute in HTTP cookies, but should not be confused with this attribute. This directive does not set the path attribute, which must be configured separately.
**Description:** Import session updates from a given HTTP response header

**Syntax:** SessionHeader *header*

**Default:** none

**Context:** server config, virtual host, directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_session

The **SessionHeader** directive defines the name of an HTTP response header which, if present, will be parsed and written to the current session.

The header value is expected to be in the URL query format, for example:

```
key1=foo&key2=&key3=bar
```

Where a key is set to the empty string, that key will be removed from the session.
**Description:** Define URL prefixes for which a session is valid

**Syntax:**

```plaintext
SessionInclude path
```

**Default:**

all URLs

**Context:**

server config, virtual host, directory, .htaccess

**Override:**

AuthConfig

**Status:**

Extension

**Module:**

mod_session

The `SessionInclude` directive allows sessions to be made valid for specific URL prefixes only. This can be used to make a website more efficient, by targeting a more precise URL space for which a session should be maintained. By default, all URLs within the directory or location are included in the session.

---

**Warning**

This directive has a similar purpose to the `path` attribute in HTTP cookies, but should not be confused with this attribute. This directive does not set the `path` attribute, which must be configured separately.
### SessionMaxAge Directive

**Description:** Define a maximum age in seconds for a session

**Syntax:**

```
SessionMaxAge maxage
```

**Default:**

```
SessionMaxAge 0
```

**Context:**

- server config
- virtual host
- directory
- `.htaccess`

**Override:**

`AuthConfig`

**Status:**

Extension

**Module:**

`mod_session`

---

The `SessionMaxAge` directive defines a time limit for which a session will remain valid. When a session is saved, this time limit is reset and an existing session can be continued. If a session becomes older than this limit without a request to the server to refresh the session, the session will time out and be removed. Where a session is used to stored user login details, this has the effect of logging the user out automatically after the given time.

Setting the maxage to zero disables session expiry.
Apache Module mod_session_cookie

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Cookie based session support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>session_cookie_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_session_cookie.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.3 and later</td>
</tr>
</tbody>
</table>

**Summary**

**Warning**

The session modules make use of HTTP cookies, and as such can fall victim to Cross Site Scripting attacks, or expose potentially private information to clients. Please ensure that the relevant risks have been taken into account before enabling the session functionality on your server.

This submodule of mod_session provides support for the storage of user sessions on the remote browser within HTTP cookies.

Using cookies to store a session removes the need for the server or a group of servers to store the session locally, or collaborate to share a session, and can be useful for high traffic environments where a server based session might be too resource intensive.

If session privacy is required, the mod_session_crypto module can be used to encrypt the contents of the session before writing the session to the client.

For more details on the session interface, see the documentation for the mod_session module.
Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod_session
mod_session_crypto
mod_session_dbd
To create a simple session and store it in a cookie called *session*, configure the session as follows:

**Browser based session**

- **Session On**
- **SessionCookieName** *session*
- **path** = /

For more examples on how the session can be configured to be read from and written to by a CGI application, see the [mod_session](http://mod_session) examples section.

For documentation on how the session can be used to store username and password details, see the [mod_auth_form](http://mod_auth_form) module.
**SessionCookieName**

<table>
<thead>
<tr>
<th>Description</th>
<th>Name and attributes for the RFC2109 cookie storing the session</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td><code>SessionCookieName name attributes</code></td>
</tr>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Context</td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td>Status</td>
<td>Extension</td>
</tr>
<tr>
<td>Module</td>
<td><code>mod_session_cookie</code></td>
</tr>
</tbody>
</table>

The **SessionCookieName** directive specifies the name and optional attributes of an RFC2109 compliant cookie inside which the session will be stored. RFC2109 cookies are set using the `Set-Cookie` HTTP header.

An optional list of cookie attributes can be specified, as per the example below. These attributes are inserted into the cookie as is, and are not interpreted by Apache. Ensure that your attributes are defined correctly as per the cookie specification.

**Cookie with attributes**

```
Session On
SessionCookieName session path=/private; domain=example.com; httpOnly; secure; version=1;
```
**Description:** Name and attributes for the RFC2965 cookie storing the session

**Syntax:** `SessionCookieName2 name attributes`

**Default:** none

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** `mod_session_cookie`

The `SessionCookieName2` directive specifies the name and optional attributes of an RFC2965 compliant cookie inside which the session will be stored. RFC2965 cookies are set using the `Set-Cookie2` HTTP header.

An optional list of cookie attributes can be specified, as per the example below. These attributes are inserted into the cookie as is, and are not interpreted by Apache. Ensure that your attributes are defined correctly as per the cookie specification.

**Cookie2 with attributes**

```
Session On
SessionCookieName2 session path=/private;domain=example.com;http
```
**Description:** Control for whether session cookies should be removed from incoming HTTP headers

**Syntax:** SessionCookieRemove On|Off

**Default:** SessionCookieRemove Off

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_session_cookie

The SessionCookieRemove flag controls whether the cookies containing the session will be removed from the headers during request processing.

In a reverse proxy situation where the Apache server acts as a server frontend for a backend origin server, revealing the contents of the session cookie to the backend could be a potential privacy violation. When set to on, the session cookie will be removed from the incoming HTTP headers.
Apache Module mod_session_crypto

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Session encryption support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>session_crypto_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_session_crypto.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.3 and later</td>
</tr>
</tbody>
</table>

**Summary**

**Warning**

The session modules make use of HTTP cookies, and as such can fall victim to Cross Site Scripting attacks, or expose potentially private information to clients. Please ensure that the relevant risks have been taken into account before enabling the session functionality on your server.

This submodule of mod_session provides support for the encryption of user sessions before being written to a local database, or written to a remote browser via an HTTP cookie.

This can help provide privacy to user sessions where the contents of the session should be kept private from the user, or where protection is needed against the effects of cross site scripting attacks.

For more details on the session interface, see the documentation for the mod_session module.
Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod_session
mod_session_cookie
mod_session_dbd
To create a simple encrypted session and store it in a cookie called `session`, configure the session as follows:

**Browser based encrypted session**

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session On</td>
<td></td>
</tr>
<tr>
<td>SessionCookieName</td>
<td><code>session</code> path=/</td>
</tr>
<tr>
<td>SessionCryptoPassphrase</td>
<td><code>secret</code></td>
</tr>
</tbody>
</table>

The session will be encrypted with the given key. Different servers can be configured to share sessions by ensuring the same encryption key is used on each server.

If the encryption key is changed, sessions will be invalidated automatically.

For documentation on how the session can be used to store username and password details, see the [mod_auth_form](#) module.
### Description:
The crypto cipher to be used to encrypt the session

### Syntax:
SessionCryptoCipher name

### Default:
aes256

### Context:
server config, virtual host, directory, .htaccess

### Status:
Experimental

### Module:
mod_session_crypto

### Compatibility:
Available in Apache 2.3.0 and later

The **SessionCryptoCipher** directive allows the cipher to be used during encryption. If not specified, the cipher defaults to aes256.

Possible values depend on the crypto driver in use, and could be one of:

- 3des192
- aes128
- aes192
- aes256
### Session Crypto Driver

**Description:** The crypto driver to be used to encrypt the session

**Syntax:**
```
SessionCryptoDriver name [param [=value]]
```

**Default:** none

**Context:** server config

**Status:** Experimental

**Module:** mod_session_crypto

**Compatibility:** Available in Apache 2.3.0 and later

The `SessionCryptoDriver` directive specifies the name of the crypto driver to be used for encryption. If not specified, the driver defaults to the recommended driver compiled into APR-util.

The **NSS** crypto driver requires some parameters for configuration, which are specified as parameters with optional values after the driver name.

**NSS without a certificate database**
```
SessionCryptoDriver nss
```

**NSS with certificate database**
```
SessionCryptoDriver nss dir=certs
```

**NSS with certificate database and parameters**
```
SessionCryptoDriver nss dir=certs key3=key3.db cert7=cert7.db
```

**NSS with paths containing spaces**
```
SessionCryptoDriver nss "dir=My Certs" key3=key3.db cert7=cert7.db
```
The NSS crypto driver might have already been configured by another part of the server, for example from mod_nss or mod_ldap. If found to have already been configured, a warning will be logged, and the existing configuration will have taken affect. To avoid this warning, use the noinit parameter as follows.

**NSS with certificate database**

```
SessionCryptoDriver nss noinit
```

To prevent confusion, ensure that all modules requiring NSS are configured with identical parameters.

The openssl crypto driver supports an optional parameter to specify the engine to be used for encryption.

**OpenSSL with engine support**

```
SessionCryptoDriver openssl engine=name
```
SessionCryptoPassphrase

**Description:** The key used to encrypt the session

**Syntax:**

```
SessionCryptoPassphrase secret [ secret ... ]
```

**Default:** none

**Context:** server config, virtual host, directory, .htaccess

**Status:** Experimental

**Module:** mod_session_crypto

**Compatibility:** Available in Apache 2.3.0 and later

The SessionCryptoPassphrase directive specifies the keys to be used to enable symmetrical encryption on the contents of the session before writing the session, or decrypting the contents of the session after reading the session.

Keys are more secure when they are long, and consist of truly random characters. Changing the key on a server has the effect of invalidating all existing sessions.

Multiple keys can be specified in order to support key rotation. The first key listed will be used for encryption, while all keys listed will be attempted for decryption. To rotate keys across multiple servers over a period of time, add a new secret to the end of the list, and once rolled out completely to all servers, remove the first key from the start of the list.

As of version 2.4.7 if the value begins with exec: the resulting command will be executed and the first line returned to standard output by the program will be used as the key.

```
#key used as-is
SessionCryptoPassphrase secret

#Run /path/to/program to get key
SessionCryptoPassphrase exec:/path/to/program
```
#Run /path/to/otherProgram and provide arguments
SessionCryptoPassphrase "exec:/path/to/otherProgram argument1"
### SessionCryptoPassphraseFile Directive

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>File containing keys used to encrypt the session</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SessionCryptoPassphraseFile <code>filename</code></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>none</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Experimental</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_session_crypto</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.3.0 and later</td>
</tr>
</tbody>
</table>

The `SessionCryptoPassphraseFile` directive specifies the name of a configuration file containing the keys to use for encrypting or decrypting the session, specified one per line. The file is read on server start, and a graceful restart will be necessary for `httpd` to pick up changes to the keys.

Unlike the `SessionCryptoPassphrase` directive, the keys are not exposed within the `httpd` configuration and can be hidden by protecting the file appropriately.

Multiple keys can be specified in order to support key rotation. The first key listed will be used for encryption, while all keys listed will be attempted for decryption. To rotate keys across multiple servers over a period of time, add a new secret to the end of the list, and once rolled out completely to all servers, remove the first key from the start of the list.
Apache Module mod_session_dbd

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>DBD/SQL based session support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>session_dbd_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_session_dbd.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache 2.3 and later</td>
</tr>
</tbody>
</table>

**Summary**

**Warning**

The session modules make use of HTTP cookies, and as such can fall victim to Cross Site Scripting attacks, or expose potentially private information to clients. Please ensure that the relevant risks have been taken into account before enabling the session functionality on your server.

This submodule of mod_session provides support for the storage of user sessions within a SQL database using the mod_dbd module.

Sessions can either be **anonymous**, where the session is keyed by a unique UUID string stored on the browser in a cookie, or **per user**, where the session is keyed against the userid of the logged in user.

SQL based sessions are hidden from the browser, and so offer a measure of privacy without the need for encryption.

Different webservers within a server farm may choose to share a database, and so share sessions with one another.

For more details on the session interface, see the documentation for the mod_session module.
Bugfix checklist

httpd changelog
Known issues
Report a bug

See also

mod_session
mod_session_crypto
mod_session_cookie
mod_dbd
Before the `mod_session_dbd` module can be configured to maintain a session, the `mod_dbd` module must be configured to make the various database queries available to the server.

There are four queries required to keep a session maintained, to select an existing session, to update an existing session, to insert a new session, and to delete an expired or empty session. These queries are configured as per the example below.

**Sample DBD configuration**

```
DBDriver psql
DBDParams "dbname=apachesession user=apache password=xxxxx host=localhost"
DBDPrepareSQL "delete from session where key = %s" deletesession
DBDPrepareSQL "update session set value = %s, expiry = %lld, key = %s where key = %s" updatesession
DBDPrepareSQL "insert into session (value, expiry, key) values (%s, %lld, %s)" insertsession
DBDPrepareSQL "select value from session where key = %s and (expiry = 0 or expiry > %lld)" selectsession
DBDPrepareSQL "delete from session where expiry != 0 and expiry < %lld" cleansession
```
Anonymous sessions are keyed against a unique UUID, and stored on the browser within an HTTP cookie. This method is similar to that used by most application servers to store session information.

To create a simple anonymous session and store it in a postgres database table called *apachesession*, and save the session ID in a cookie called *session*, configure the session as follows:

```
SQL based anonymous session
Session On
SessionDBDCookieName session path=/
```

For more examples on how the session can be configured to be read from and written to by a CGI application, see the *mod_session* examples section.

For documentation on how the session can be used to store username and password details, see the *mod auth form* module.
Per user sessions are keyed against the username of a successfully authenticated user. It offers the most privacy, as no external handle to the session exists outside of the authenticated realm.

Per user sessions work within a correctly configured authenticated environment, be that using basic authentication, digest authentication or SSL client certificates. Due to the limitations of who came first, the chicken or the egg, per user sessions cannot be used to store authentication credentials from a module like `mod_auth_form`.

To create a simple per user session and store it in a postgres database table called `apachesession`, and with the session keyed to the userid, configure the session as follows:

**SQL based per user session**

```
Session On
SessionDBDPerUser On
```
Over the course of time, the database can be expected to start accumulating expired sessions. At this point, the mod_session_dbd module is not yet able to handle session expiry automatically.

**Warning**

The administrator will need to set up an external process via cron to clean out expired sessions.
**Description:** Name and attributes for the RFC2109 cookie storing the session ID

**Syntax:** SessionDBDCookieName name attributes

**Default:** none

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_session_dbd

The `SessionDBDCookieName` directive specifies the name and optional attributes of an RFC2109 compliant cookie inside which the session ID will be stored. RFC2109 cookies are set using the `Set-Cookie` HTTP header.

An optional list of cookie attributes can be specified, as per the example below. These attributes are inserted into the cookie as is, and are not interpreted by Apache. Ensure that your attributes are defined correctly as per the cookie specification.

**Cookie with attributes**

```
Session On
SessionDBDCookieName session path=/private;domain=example.com;httponly;secure;version=1;
```
**Description:** Name and attributes for the RFC2965 cookie storing the session ID

**Syntax:** SessionDBDCookieName2 name attributes

**Default:** none

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_session_dbd

The `SessionDBDCookieName2` directive specifies the name and optional attributes of an RFC2965 compliant cookie inside which the session ID will be stored. RFC2965 cookies are set using the `Set-Cookie2` HTTP header.

An optional list of cookie attributes can be specified, as per the example below. These attributes are inserted into the cookie as is, and are not interpreted by Apache. Ensure that your attributes are defined correctly as per the cookie specification.

### Cookie with attributes

```
Session On
SessionDBDCookieName2 session path=/private;domain=example.com
```
**Description:** Control for whether session ID cookies should be removed from incoming HTTP headers

**Syntax:** SessionDBDCookieRemove On|Off

**Default:** SessionDBDCookieRemove On

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_session_dbd

The **SessionDBDCookieRemove** flag controls whether the cookies containing the session ID will be removed from the headers during request processing.

In a reverse proxy situation where the Apache server acts as a server frontend for a backend origin server, revealing the contents of the session ID cookie to the backend could be a potential privacy violation. When set to on, the session ID cookie will be removed from the incoming HTTP headers.
**Description:** The SQL query to use to remove sessions from the database

**Syntax:** SessionDBDDeleteLabel *label*

**Default:** SessionDBDDeleteLabel deletesession

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_session_dbd

The `SessionDBDDeleteLabel` directive sets the default delete query label to be used to delete an expired or empty session. This label must have been previously defined using the `DBDPrepareSQL` directive.
**Description:** The SQL query to use to insert sessions into the database

**Syntax:** SessionDBDInsertLabel *label*

**Default:** SessionDBDInsertLabel insertsession

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_session_dbd

The `SessionDBDInsertLabel` directive sets the default insert query label to be used to load in a session. This label must have been previously defined using the `DBDPrepareSQL` directive.

If an attempt to update the session affects no rows, this query will be called to insert the session into the database.
**Description:** Enable a per user session

**Syntax:** SessionDBDPerUser On|Off

**Default:** SessionDBDPerUser Off

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_session_dbd

The `SessionDBDPerUser` flag enables a per user session keyed against the user's login name. If the user is not logged in, this directive will be ignored.
**Description:** The SQL query to use to select sessions from the database

**Syntax:** SessionDBDSelectLabel *label*

**Default:** SessionDBDSelectLabel selectsession

**Context:** server config, virtual host, directory, .htaccess

**Status:** Extension

**Module:** mod_session_dbd

The `SessionDBDSelectLabel` directive sets the default select query label to be used to load in a session. This label must have been previously defined using the `DBDPrepareSQL` directive.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>The SQL query to use to update existing sessions in the database</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SessionDBDUpdateLabel <em>label</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>SessionDBDUpdateLabel updatesession</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_session_dbd</td>
</tr>
</tbody>
</table>

The `SessionDBDUpdateLabel` directive sets the default update query label to be used to load in a session. This label must have been previously defined using the `DBDPrepareSQL` directive.

If an attempt to update the session affects no rows, the insert query will be called to insert the session into the database. If the database supports InsertOrUpdate, override this query to perform the update in one query instead of two.
mod_setenvif

BrowserMatch ^Mozilla netscape
BrowserMatch MSIE !netscape

Bugfix checklist

httpd changelog
Known issues
Report a bug

Apache
BrowserMatch  SetEnvIf  User-Agent HT

BrowserMatchNoCase Robot is_a_robot
SetEnvIfNoCase User-Agent Robot is_a_robot

BrowserMatch ^Mozilla forms jpeg=yes browser=netscape
BrowserMatch "^Mozilla/[2-3]" tables agif frames javascript
BrowserMatch MSIE !javascript
HTTP User-Agent

BrowserMatchNoCase regex [!]env-variable[=value] [[!]env-variable[=value]]
...

, , .htaccess

FileInfo

Base

mod_setenvif

BrowserMatchNoCase

BrowserMatch

BrowserMatchNoCase mac platform=macintosh
BrowserMatchNoCase win platform=windows

BrowserMatch

BrowserMatchNoCase

SetEnvIfNoCase

2:

BrowserMatchNoCase Robot is_a_robot
SetEnvIfNoCase User-Agent Robot is_a_robot
SetEnvIf

SetEnvIf attribute regex [!]env-variable[=value] [![env-variable][=value]]
...,
,,.htaccess
\_ FileInfo
\_ Base
\_ mod_setenvif

**SetEnvIf**

1. HTTP (RFC 2616) Host, User-Agent, Referer, Accept-Language

2. :
   - Remote_Host - ()
   - Remote_Addr - IP
   - Server_Addr - IP (2.0.43)
   - Request_Method - (GET, POST)
   - Request_Protocol -
   - Request_URI - URL

3. **SetEnvIf** SetEnvIf[NoCase]()

4. SSL oid ID
   - oid
   (regex) POSIX.2 grep regex attribute
1. `varname`
2. `!varname`
3. `varname=value`

"1"

regex

: 
SetEnvIf Request_URI "\.gif$" object_is_image=gif
SetEnvIf Request_URI "\.jpg$" object_is_image=jpg
SetEnvIf Request_URI "\.xbm$" object_is_image=xbm:
SetEnvIf Referer www\ mydomain\ example\ com intra_site_referral:
SetEnvIf object_is_image xbm XBIT_PROCESSING=1:
SetEnvIf OID("2.16.840.1.113730.1.13") "(\.*"
NetscapeComment=$1:
SetEnvIf ^TS* ^[a-z].* HAVE_TS

object_is_image
intra_site_referral

6 NetscapeComment SSL

"TS" [a-z]

- Apache
Sets environment variables based on an ap_expr expression.

Base

.mod_setenvif

Documentation not yet translated. Please see English version of document.
SetEnvIfNoCase attribute regex [!]env-variable[=value] [!]env-variable[=value]
...
, , , .htaccess
FileInfo
Base
mod_setenvif

SetEnvIfNoCase Host Apache\.Org site=apache

HTTP Host: Apache\.Org apache.org
site " apache"
Apache Module mod_slotmem_plain

**Description:** Slot-based shared memory provider.

**Status:** Extension

**Module Identifier:** slotmem_plain_module

**Source File:** mod_slotmem_plain.c

**Summary**

mod_slotmem_plain is a memory provider which provides for creation and access to a plain memory segment in which the datasets are organized in "slots."

If the memory needs to be shared between threads and processes, a better provider would be **mod_slotmem_shm**.

mod_slotmem_plain provides the following API functions:

- **apr_status_t doall(ap_slotmem_instance_t *s,**
  ap_slotmem_callback_fn_t *func, void *data, apr_pool_t *pool)**
  call the callback on all worker slots

- **apr_status_t create(ap_slotmem_instance_t **new,**
  const char *name,**
  apr_size_t item_size,**
  unsigned int item_num,**
  apr_slotmem_type_t type,**
  apr_pool_t *pool)**
  create a new slotmem with each item size is item_size.

- **apr_status_t attach(ap_slotmem_instance_t **new,**
  const char *name,**
  apr_size_t *item_size,**
  unsigned int *item_num,**
  apr_pool_t *pool)**
  attach to an existing slotmem.

- **apr_status_t dpotr(ap_slotmem_instance_t *s,**
  unsigned int item_id,**
  void**mem)**
  get the direct pointer to the memory associated with this worker slot.
apr_status_t get(ap_slotmem_instance_t *s, unsigned int item_id, unsigned char *dest, apr_size_t dest_len)
    get/read the memory from this slot to dest

apr_status_t put(ap_slotmem_instance_t *slot, unsigned int item_id, unsigned char *src, apr_size_t src_len)
    put/write the data from src to this slot

unsigned int num_slots(ap_slotmem_instance_t *s)
    return the total number of slots in the segment

apr_size_t slot_size(ap_slotmem_instance_t *s)
    return the total data size, in bytes, of a slot in the segment

apr_status_t grab(ap_slotmem_instance_t *s, unsigned int *item_id);
    grab or allocate the first free slot and mark as in-use (does not do any data copying)

apr_status_t fgrab(ap_slotmem_instance_t *s, unsigned int item_id);
    forced grab or allocate the specified slot and mark as in-use (does not do any data copying)

apr_status_t release(ap_slotmem_instance_t *s, unsigned int item_id);
    release or free a slot and mark as not in-use (does not do any data copying)
### Apache Module mod_slotmem_shm

<table>
<thead>
<tr>
<th>Description:</th>
<th>Slot-based shared memory provider.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>slotmem_shm_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_slotmem_shm.c</td>
</tr>
</tbody>
</table>

**Summary**

mod_slotmem_shm is a memory provider which provides for creation and access to a shared memory segment in which the datasets are organized in "slots."

All shared memory is cleared and cleaned with each restart, whether graceful or not. The data itself is stored and restored within a file noted by the name parameter in the create and attach calls. If not specified with an absolute path, the file will be created relative to the path specified by the `DefaultRuntimeDir` directive.

mod_slotmem_shm provides the following API functions:

- `apr_status_t doall(ap_slotmem_instance_t *s, ap_slotmem_callback_fn_t *func, void *data, apr_pool_t *pool)`
  
  call the callback on all worker slots

- `apr_status_t create(ap_slotmem_instance_t **new, const char *name, apr_size_t item_size, unsigned int item_num, ap_slotmem_type_t type, apr_pool_t *pool)`

  create a new slotmem with each item size is item_size. name is used to generate a filename for the persistent store of the shared memory if configured. Values are:

  "none"

    Anonymous shared memory and no persistent store
 apr_status_t attach(ap_slotmem_instance_t **new, const char *name, apr_size_t *item_size, unsigned int *item_num, apr_pool_t *pool)
    attach to an existing slotmem. See create for description of name parameter.

 apr_status_t dptr(ap_slotmem_instance_t *s, unsigned int item_id, void**mem)
    get the direct pointer to the memory associated with this worker slot.

 apr_status_t get(ap_slotmem_instance_t *s, unsigned int item_id, unsigned char *dest, apr_size_t dest_len)
    get/read the memory from this slot to dest

 apr_status_t put(ap_slotmem_instance_t *slot, unsigned int item_id, unsigned char *src, apr_size_t src_len)
    put/write the data from src to this slot

 unsigned int num_slots(ap_slotmem_instance_t *s)
    return the total number of slots in the segment

 apr_size_t slot_size(ap_slotmem_instance_t *s)
    return the total data size, in bytes, of a slot in the segment

 apr_status_t grab(ap_slotmem_instance_t *s, unsigned int *item_id);
    grab or allocate the first free slot and mark as in-use (does not do any data copying)

 apr_status_t fgrab(ap_slotmem_instance_t *s, unsigned int item_id);
    forced grab or allocate the specified slot and mark as in-use (does not do any data copying)
apr_status_t release(ap_slotmem_instance_t *s, unsigned int item_id);

release or free a slot and mark as not in-use (does not do any data copying)
Apache mod_so

- Extension
- so_module
- mod_so.c
- Window ()
  Base

Unix

Apache HTTP Server
Windows .dll Apache httpd mod_whatever PHP

mod_so ApacheModuleFoo.dll

 Apache httpd API UNIX Windows
 Windows Unix

 Unix

Configure

 DLL

Apache httpd

 DLL

 AP_MODULE_DECLARE_DATA (Apache httpd )

 module foo_module;

 module AP_MODULE_DECLARE_DATA foo_module;

 Unix Windows

 DLL libhttpd.lib

 DLL modules
LoadFile

LoadFile libexec/libxmlparse.so
LoadModule module filename
.
Extension
mod_so

LoadModule  filename

LoadModule status_module modules/mod_status

ServerRoot modules
Apache Module mod_socache_dbm

<table>
<thead>
<tr>
<th>Description:</th>
<th>DBM based shared object cache provider.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Extension</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>socache_dbm_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_socache_dbm.c</td>
</tr>
</tbody>
</table>

Summary

mod_socache_dbm is a shared object cache provider which provides for creation and access to a cache backed by a DBM database.

dbm:/path/to/datafile

Details of other shared object cache providers can be found here.
# Apache Module mod_socache_dc

<table>
<thead>
<tr>
<th>Description:</th>
<th>Distcache based shared object cache provider.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Extension</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>socache_dc_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_socache_dc.c</td>
</tr>
</tbody>
</table>

## Summary

`mod_socache_dc` is a shared object cache provider which provides for creation and access to a cache backed by the [distcache](#) distributed session caching libraries.

Details of other shared object cache providers can be found [here](#).

---

Copyright 2017 The Apache Software Foundation.
Licensed under the [Apache License, Version 2.0](#).

[Modules](#) | [Directives](#) | [FAQ](#) | [Glossary](#) | [Sitemap](#)
Apache Module mod_socache_memcache

<table>
<thead>
<tr>
<th>Description:</th>
<th>Memcache based shared object cache provider.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Extension</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>socache_memcache_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_socache_memcache.c</td>
</tr>
</tbody>
</table>

Summary

mod_socache_memcache is a shared object cache provider which provides for creation and access to a cache backed by the memcached high-performance, distributed memory object caching system.

This shared object cache provider's "create" method requires a comma separated list of memcached host/port specifications. If using this provider via another modules configuration (such as SSLSessionCache), provide the list of servers as the optional "arg" parameter.

```
SSLSessionCache memcache:memcache.example.com:12345,memcache2.example.com:12345
```

Details of other shared object cache providers can be found [here](#).
**Description:** Keepalive time for idle connections

**Syntax:** `MemcacheConnTTL num[units]`

**Default:** `MemcacheConnTTL 15s`

**Context:** server config, virtual host

**Status:** Extension

**Module:** `mod_socache_memcache`

**Compatibility:** Available in Apache 2.4.17 and later

Set the time to keep idle connections with the memcache server(s) alive (threaded platforms only).

Valid values for `MemcacheConnTTL` are times up to one hour. 0 means no timeout.

This timeout defaults to units of seconds, but accepts suffixes for milliseconds (ms), seconds (s), minutes (min), and hours (h).

Before Apache 2.4.17, this timeout was hardcoded and its value was 600 usec. So, the closest configuration to match the legacy behaviour is to set `MemcacheConnTTL` to 1ms.

```sh
# Set a timeout of 10 minutes
MemcacheConnTTL 10min
# Set a timeout of 60 seconds
MemcacheConnTTL 60
```
Apache Module mod_socache_shmcb

**Description:** shmcb based shared object cache provider.

**Status:** Extension

**Module Identifier:** socache_shmcb_module

**Source File:** mod_socache_shmcb.c

### Summary

mod_socache_shmcb is a shared object cache provider which provides for creation and access to a cache backed by a high-performance cyclic buffer inside a shared memory segment.

shmcb:/path/to/datafile(512000)

Details of other shared object cache providers can be found [here](#).
### Apache mod_speling

<table>
<thead>
<tr>
<th>URL</th>
<th>Extension</th>
<th>speling_module</th>
<th>mod_speling.c</th>
</tr>
</thead>
</table>

Apache
CheckCaseOnly on | off
CheckCaseOnly Off

Options
Extension
mod_speling
spelling
CheckSpelling on/off
CheckSpelling Off
.

CheckSpelling Apache 1.1  Apache 1.3
Apache  Apache 1.3.2

DAV mod_speling

(doc34.html)

(Location /st

(http://my.host/~apahce/) )
### Apache Module mod_ssl

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Strong cryptography using the Secure Sockets Layer (SSL) and Transport Layer Security (TLS) protocols</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>ssl_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_ssl.c</td>
</tr>
</tbody>
</table>

**Summary**

This module provides SSL v3 and TLS v1.x support for the Apache HTTP Server. SSL v2 is no longer supported.

This module relies on [OpenSSL](https://www.openssl.org) to provide the cryptography engine.

Further details, discussion, and examples are provided in the [SSL documentation](https://httpd.apache.org/docs/current/ssl/ssl_intro.html).
This module can be configured to provide several items of SSL information as additional environment variables to the SSI and CGI namespace. This information is not provided by default for performance reasons. (See SSLOptions StdEnvVars, below.) The generated variables are listed in the table below. For backward compatibility the information can be made available under different names, too. Look in the Compatibility chapter for details on the compatibility variables.

<table>
<thead>
<tr>
<th>Variable Name:</th>
<th>Value Type:</th>
<th>Description:</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTPS</td>
<td>flag</td>
<td>HTTPS is being used</td>
</tr>
<tr>
<td>SSL_PROTOCOL</td>
<td>string</td>
<td>The SSL protocol version (SSLv3, TLSv1, TLSv1.1, TLSv1.2)</td>
</tr>
<tr>
<td>SSL_SESSION_ID</td>
<td>string</td>
<td>The hex-encoded session id</td>
</tr>
<tr>
<td>SSL_SESSION_RESUMED</td>
<td>string</td>
<td>Initial or Resumed Session. Note: multiple requests may be over the same (Initial or Resumed) SSL session if HTTP KeepAlive is in use</td>
</tr>
<tr>
<td>SSL_SECURE_RENEG</td>
<td>string</td>
<td>true if secure renegotiation is supported, else false</td>
</tr>
<tr>
<td>SSL_CIPHER</td>
<td>string</td>
<td>The cipher specification name</td>
</tr>
<tr>
<td>SSL_CIPHER_EXPORT</td>
<td>string</td>
<td>true if cipher is an export cipher</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>----------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SSL_CIPHER_USEKEYSIZE</td>
<td>number</td>
<td>Number of cipher bits used (actually used)</td>
</tr>
<tr>
<td>SSL_CIPHER_ALGKEYSIZE</td>
<td>number</td>
<td>Number of cipher bits possible</td>
</tr>
<tr>
<td>SSL_COMPRESS_METHOD</td>
<td>string</td>
<td>SSL compression method negotiated</td>
</tr>
<tr>
<td>SSL_VERSION_INTERFACE</td>
<td>string</td>
<td>The mod_ssl program version</td>
</tr>
<tr>
<td>SSL_VERSION_LIBRARY</td>
<td>string</td>
<td>The OpenSSL program version</td>
</tr>
<tr>
<td>SSL_CLIENT_M_VERSION</td>
<td>string</td>
<td>The version of the client certificate</td>
</tr>
<tr>
<td>SSL_CLIENT_M_SERIAL</td>
<td>string</td>
<td>The serial of the client certificate</td>
</tr>
<tr>
<td>SSL_CLIENT_S_DN</td>
<td>string</td>
<td>Subject DN in client certificate</td>
</tr>
<tr>
<td>SSL_CLIENT_S_DN_x509</td>
<td>string</td>
<td>Component of client Subject DN</td>
</tr>
<tr>
<td>SSL_CLIENT_SAN_Email_n</td>
<td>string</td>
<td>Client certificate's subjectAltName extension entries of type rfc822Name</td>
</tr>
<tr>
<td>SSL_CLIENT_SAN_DNS_n</td>
<td>string</td>
<td>Client certificate's subjectAltName extension entries of type dNSName</td>
</tr>
<tr>
<td>SSL_CLIENT_SAN_OTHER_msUPN_n</td>
<td>string</td>
<td>Client certificate's subjectAltName extension entries of type otherName, Microsoft User Principal Name form (OID</td>
</tr>
<tr>
<td>Variable Name</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SSL_CLIENT_I_DN</td>
<td>string</td>
<td>Issuer DN of client's certificate</td>
</tr>
<tr>
<td>SSL_CLIENT_I_DN_x509</td>
<td>string</td>
<td>Component of client's Issuer DN</td>
</tr>
<tr>
<td>SSL_CLIENT_V_START</td>
<td>string</td>
<td>Validity of client's certificate (start time)</td>
</tr>
<tr>
<td>SSL_CLIENT_V_END</td>
<td>string</td>
<td>Validity of client's certificate (end time)</td>
</tr>
<tr>
<td>SSL_CLIENT_V_REMAIN</td>
<td>string</td>
<td>Number of days until client's certificate expires</td>
</tr>
<tr>
<td>SSL_CLIENT_A_SIG</td>
<td>string</td>
<td>Algorithm used for signature of client's certificate</td>
</tr>
<tr>
<td>SSL_CLIENT_A_KEY</td>
<td>string</td>
<td>Algorithm used for public key of client's certificate</td>
</tr>
<tr>
<td>SSL_CLIENT_CERT</td>
<td>string</td>
<td>PEM-encoded client certificate</td>
</tr>
<tr>
<td>SSL_CLIENT_CERT_CHAIN_n</td>
<td>string</td>
<td>PEM-encoded certificates in client certificate chain</td>
</tr>
<tr>
<td>SSL_CLIENT_CERTRFC4523_CEA</td>
<td>string</td>
<td>Serial number and issuer of the certificate. Format matches that of the CertificateExactAssertion in RFC4523</td>
</tr>
<tr>
<td>SSL_CLIENT_VERIFY</td>
<td>string</td>
<td>NONE, SUCCESS, GENEROUS or FAILED: reason</td>
</tr>
<tr>
<td>SSL_SERVER_M_VERSION</td>
<td>string</td>
<td>The version of the server</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SSL_SERVER_M_SERIAL</td>
<td>string</td>
<td>The serial of the server certificate</td>
</tr>
<tr>
<td>SSL_SERVER_S_DN</td>
<td>string</td>
<td>Subject DN in server certificate</td>
</tr>
<tr>
<td>SSL_SERVER_SAN_Email_n</td>
<td>string</td>
<td>Server certificate's subjectAltName extension entries rfc822Name</td>
</tr>
<tr>
<td>SSL_SERVER_SAN_DNS_n</td>
<td>string</td>
<td>Server certificate's subjectAltName extension entries dNSName</td>
</tr>
<tr>
<td>SSL_SERVER_SAN_OTHER_dnsSRV_n</td>
<td>string</td>
<td>Server certificate's subjectAltName extension entries otherName, SRV form (OID 1.3.6.1.5.5.7.8.7, 4985)</td>
</tr>
<tr>
<td>SSL_SERVER_S_DN_x509</td>
<td>string</td>
<td>Component of server's Subject DN</td>
</tr>
<tr>
<td>SSL_SERVER_I_DN</td>
<td>string</td>
<td>Issuer DN of server certificate</td>
</tr>
<tr>
<td>SSL_SERVER_I_DN_x509</td>
<td>string</td>
<td>Component of server's Issuer DN</td>
</tr>
<tr>
<td>SSL_SERVER_V_START</td>
<td>string</td>
<td>Validity of server's certificate (start time)</td>
</tr>
<tr>
<td>SSL_SERVER_V_END</td>
<td>string</td>
<td>Validity of server's certificate (end time)</td>
</tr>
<tr>
<td>SSL_SERVER_A_SIG</td>
<td>string</td>
<td>Algorithm used for signature of server certificate</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------</td>
<td>-------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SSL_SERVER_A_KEY</td>
<td>string</td>
<td>Algorithm used for the public key of server's certificate</td>
</tr>
<tr>
<td>SSL_SERVER_CERT</td>
<td>string</td>
<td>PEM-encoded server certificate</td>
</tr>
<tr>
<td>SSL_SRP_USER</td>
<td>string</td>
<td>SRP username</td>
</tr>
<tr>
<td>SSL_SRP_USERINFO</td>
<td>string</td>
<td>SRP user info</td>
</tr>
<tr>
<td>SSL_TLS_SNI</td>
<td>string</td>
<td>Contents of the SNI TLS extension (if supplied with ClientHello)</td>
</tr>
</tbody>
</table>

`x509` specifies a component of an X.509 DN; one of C, ST, L, O, OU, CN, T, I, G, S, D, UID, Email. In Apache 2.1 and later, `x509` may also include a numeric `_n` suffix. If the DN in question contains multiple attributes of the same name, this suffix is used as a zero-based index to select a particular attribute. For example, where the server certificate subject DN included two OU attributes, `SSL_SERVER_S_DN_OU_0` and `SSL_SERVER_S_DN_OU_1` could be used to reference each. A variable name without a `_n` suffix is equivalent to that name with a `_0` suffix; the first (or only) attribute. When the environment table is populated using the `StdEnvVars` option of the `SSLOptions` directive, the first (or only) attribute of any DN is added only under a non-suffixed name; i.e. no `_0` suffixed entries are added.

The format of the `*_DN` variables has changed in Apache HTTPD 2.3.11. See the `LegacyDNStringFormat` option for `SSLOptions` for details.

`SSL_CLIENT_V_REMAIN` is only available in version 2.1 and later.

A number of additional environment variables can also be used in
SSLRequire expressions, or in custom log formats:

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Log Field</th>
<th>Log Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_USER_AGENT</td>
<td>PATH_INFO</td>
<td>AUTH_TYPE</td>
</tr>
<tr>
<td>HTTP_REFERER</td>
<td>QUERY_STRING</td>
<td>SERVER_SOFTWARE</td>
</tr>
<tr>
<td>HTTP_COOKIE</td>
<td>REMOTE_HOST</td>
<td>API_VERSION</td>
</tr>
<tr>
<td>HTTP_FORWARDED</td>
<td>REMOTE_IDENT</td>
<td>TIME_YEAR</td>
</tr>
<tr>
<td>HTTP_HOST</td>
<td>IS_SUBREQ</td>
<td>TIME_MON</td>
</tr>
<tr>
<td>HTTP_PROXY_CONNECTION</td>
<td>DOCUMENT_ROOT</td>
<td>TIME_DAY</td>
</tr>
<tr>
<td>HTTP_ACCEPT</td>
<td>SERVER_ADMIN</td>
<td>TIME_HOUR</td>
</tr>
<tr>
<td>THE_REQUEST</td>
<td>SERVER_NAME</td>
<td>TIME_MIN</td>
</tr>
<tr>
<td>REQUEST_FILENAME</td>
<td>SERVER_PORT</td>
<td>TIME_SEC</td>
</tr>
<tr>
<td>REQUEST_METHOD</td>
<td>SERVER_PROTOCOL</td>
<td>TIME_WDAY</td>
</tr>
<tr>
<td>REQUEST_SCHEME</td>
<td>REMOTE_ADDR</td>
<td>TIME</td>
</tr>
<tr>
<td>REQUEST_URI</td>
<td>REMOTE_USER</td>
<td></td>
</tr>
</tbody>
</table>

In these contexts, two special formats can also be used:

**ENV:** `variblename`  
This will expand to the standard environment variable `variblename`.

**HTTP:** `headername`  
This will expand to the value of the request header with name `headername`.  

⚠️
When **mod_ssl** is built into Apache or at least loaded (under DSO situation) additional functions exist for the Custom Log Format of **mod_log_config**. First there is an additional ``%{varname}x`` eXtension format function which can be used to expand any variables provided by any module, especially those provided by mod_ssl which can you find in the above table.

For backward compatibility there is additionally a special ``%{name}c`` cryptography format function provided. Information about this function is provided in the Compatibility chapter.

**Example**

```
CustomLog "logs/ssl_request_log" "%t %h %{SSL_PROTOCOL}x %{SSL_"
```

These formats even work without setting the StdEnvVars option of theSSLOptions directive.
mod_ssl sets "notes" for the request which can be used in logging with the %{name}n format string in mod_log_config.

The notes supported are as follows:

**ssl-access-forbidden**
This note is set to the value 1 if access was denied due to an SSLRequire or SSLRequireSSL directive.

**ssl-secure-reneg**
If mod_ssl is built against a version of OpenSSL which supports the secure renegotiation extension, this note is set to the value 1 if SSL is in use for the current connection, and the client also supports the secure renegotiation extension. If the client does not support the secure renegotiation extension, the note is set to the value 0. If mod_ssl is not built against a version of OpenSSL which supports secure renegotiation, or if SSL is not in use for the current connection, the note is not set.
When mod_ssl is built into Apache or at least loaded (under DSO situation) any variables provided by mod_ssl can be used in expressions for the ap_expr Expression Parser. The variables can be referenced using the syntax ``%{varname}``. Starting with version 2.4.18 one can also use the mod_rewrite style syntax ``%{SSL:varname}`" or the function style syntax ``ssl(varname)".

**Example (using mod_headers)**

```
Header set X-SSL-PROTOCOL "expr=%{SSL_PROTOCOL}"
Header set X-SSL-CIPHER "expr=%{SSL:SSL_CIPHER}"
```

This feature even works without setting the StdEnvVars option of the SSLOptions directive.
mod_ssl provides a few authentication providers for use with mod_authz_core's Require directive.

**Require ssl**
The ssl provider denies access if a connection is not encrypted with SSL. This is similar to the SSLRequireSSL directive.

```
Require ssl
```

**Require ssl-verify-client**
The ssl provider allows access if the user is authenticated with a valid client certificate. This is only useful if SSLVerifyClient optional is in effect.

The following example grants access if the user is authenticated either with a client certificate or by username and password.

```
Require ssl-verify-client
Require valid-user
```
**Description:** File of concatenated PEM-encoded CA Certificates for Client Auth

**Syntax:** `SSLCACertificateFile file-path`

**Context:** server config, virtual host

**Status:** Extension

**Module:** `mod_ssl`

This directive sets the *all-in-one* file where you can assemble the Certificates of Certification Authorities (CA) whose clients you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to `SSLCACertificatePath`.

**Example**

```
SSLCACertificateFile "/usr/local/apache2/conf/ssl.crt/ca-bundle-client.crt"
```
**Description:** Directory of PEM-encoded CA Certificates for Client Auth

**Syntax:**
SSLCACertificatePath *directory-path*

**Context:**
server config, virtual host

**Status:**
Extension

**Module:**
mod_ssl

This directive sets the directory where you keep the Certificates of Certification Authorities (CAs) whose clients you deal with. These are used to verify the client certificate on Client Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named *hash-value*.N. And you should always make sure this directory contains the appropriate symbolic links.

**Example**
SSLCACertificatePath "/usr/local/apache2/conf/ssl.crt/"
**Description:** File of concatenated PEM-encoded CA Certificates for defining acceptable CA names

**Syntax:** SSLCADNRequestFile file-path

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

When a client certificate is requested by mod_ssl, a list of acceptable Certificate Authority names is sent to the client in the SSL handshake. These CA names can be used by the client to select an appropriate client certificate out of those it has available.

If neither of the directives SSLCADNRequestPath or SSLCADNRequestFile are given, then the set of acceptable CA names sent to the client is the names of all the CA certificates given by the SSLCACertificateFile and SSLCACertificatePath directives; in other words, the names of the CAs which will actually be used to verify the client certificate.

In some circumstances, it is useful to be able to send a set of acceptable CA names which differs from the actual CAs used to verify the client certificate - for example, if the client certificates are signed by intermediate CAs. In such cases, SSLCADNRequestPath and/or SSLCADNRequestFile can be used; the acceptable CA names are then taken from the complete set of certificates in the directory and/or file specified by this pair of directives.

**Example**

SSLCADNRequestFile must specify an all-in-one file containing a concatenation of PEM-encoded CA certificates.
SSLCA DNRequestFile "/usr/local/apache2/conf/ca-names.crt"
Description: Directory of PEM-encoded CA Certificates for defining acceptable CA names

Syntax: SSLCADNRequestPath directory-path

Context: server config, virtual host

Status: Extension

Module: mod_ssl

This optional directive can be used to specify the set of acceptable CA names which will be sent to the client when a client certificate is requested. See the SSLCADNRequestFile directive for more details.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named hash-value.N. And you should always make sure this directory contains the appropriate symbolic links.

Example

SSLCADNRequestPath "/usr/local/apache2/conf/ca-names.crt/"
SSLCARevocationCheck

**Description:** Enable CRL-based revocation checking

**Syntax:** SSLCARevocationCheck chain|leaf|none

**Default:** SSLCARevocationCheck none

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Optional flags available in httpd 2.4.21 or later

Enables certificate revocation list (CRL) checking. At least one of SSLCARevocationFile or SSLCARevocationPath must be configured. When set to chain (recommended setting), CRL checks are applied to all certificates in the chain, while setting it to leaf limits the checks to the end-entity cert.

The available flags are:

- **no_crl_for_cert_ok**
  Prior to version 2.3.15, CRL checking in mod_ssl also succeeded when no CRL(s) for the checked certificate(s) were found in any of the locations configured with SSLCARevocationFile or SSLCARevocationPath.

With the introduction of SSLCARevocationFile, the behavior has been changed: by default with chain or leaf, CRLs must be present for the validation to succeed - otherwise it will fail with an "unable to get certificate CRL" error.

The flag no_crl_for_cert_ok allows to restore previous behaviour.
Example
SSLCARevocationCheck chain

Compatibility with versions 2.2
SSLCARevocationCheck chain no_crl_for_cert_ok
**Description:** File of concatenated PEM-encoded CA CRLs for Client Auth

**Syntax:** SSLCARevocationFile file-path

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificate Revocation Lists (CRL) of Certification Authorities (CA) whose clients you deal with. These are used for Client Authentication. Such a file is simply the concatenation of the various PEM-encoded CRL files, in order of preference. This can be used alternatively and/or additionally to SSLCARevocationPath.

**Example**

SSLCARevocationFile "/usr/local/apache2/conf/ssl.crl/ca-bundle-"
**Description:** Directory of PEM-encoded CA CRLs for Client Auth

**Syntax:**
SSLCARevocationPath *directory-path*

**Context:**
server config, virtual host

**Status:**
Extension

**Module:**
mod_ssl

This directive sets the directory where you keep the Certificate Revocation Lists (CRL) of Certification Authorities (CAs) whose clients you deal with. These are used to revoke the client certificate on Client Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you have not only to place the CRL files there. Additionally you have to create symbolic links named *hash-value*.rN. And you should always make sure this directory contains the appropriate symbolic links.

**Example**
SSLCARevocationPath "/usr/local/apache2/conf/ssl.crl/"
SSLCertificateChainFile

**Description:** File of PEM-encoded Server CA Certificates

**Syntax:** SSLCertificateChainFile *file-path*

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

---

**SSLCertificateChainFile is deprecated**

SSLCertificateChainFile became obsolete with version 2.4.8, when SSLCertificateFile was extended to also load intermediate CA certificates from the server certificate file.

This directive sets the optional *all-in-one* file where you can assemble the certificates of Certification Authorities (CA) which form the certificate chain of the server certificate. This starts with the issuing CA certificate of the server certificate and can range up to the root CA certificate. Such a file is simply the concatenation of the various PEM-encoded CA Certificate files, usually in certificate chain order.

This should be used alternatively and/or additionally to SSLCACertificatePath for explicitly constructing the server certificate chain which is sent to the browser in addition to the server certificate. It is especially useful to avoid conflicts with CA certificates when using client authentication. Because although placing a CA certificate of the server certificate chain into SSLCACertificatePath has the same effect for the certificate chain construction, it has the side-effect that client certificates issued by this same CA certificate are also accepted on client authentication.

But be careful: Providing the certificate chain works only if you are using a *single* RSA or DSA based server certificate. If you are
using a coupled RSA+DSA certificate pair, this will work only if actually both certificates use the *same* certificate chain. Else the browsers will be confused in this situation.

**Example**

SSLCertificateChainFile "/usr/local/apache2/conf/ssl.crt/ca.crt"
**Description:** Server PEM-encoded X.509 certificate data file

**Syntax:** SSLCertificateFile *file-path*

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive points to a file with certificate data in PEM format. At a minimum, the file must include an end-entity (leaf) certificate. The directive can be used multiple times (referencing different filenames) to support multiple algorithms for server authentication - typically RSA, DSA, and ECC. The number of supported algorithms depends on the OpenSSL version being used for mod_ssl: with version 1.0.0 or later, openssl list-public-key-algorithms will output a list of supported algorithms, see also the note below about limitations of OpenSSL versions prior to 1.0.2 and the ways to work around them.

The files may also include intermediate CA certificates, sorted from leaf to root. This is supported with version 2.4.8 and later, and obsoletes SSLCertificateChainFile. When running with OpenSSL 1.0.2 or later, this allows to configure the intermediate CA chain on a per-certificate basis.

Custom DH parameters and an EC curve name for ephemeral keys, can also be added to end of the first file configured using SSLCertificateFile. This is supported in version 2.4.7 or later. Such parameters can be generated using the commands openssl dhparam and openssl ecparam. The parameters can be added as-is to the end of the first certificate file. Only the first file can be used for custom parameters, as they are applied independently of the authentication algorithm type.

Finally the end-entity certificate's private key can also be added to
the certificate file instead of using a separate
$\texttt{SSLCertificateKeyFile}$ directive. This practice is highly
discouraged. If it is used, the certificate files using such an
embedded key must be configured after the certificates using a
separate key file. If the private key is encrypted, the pass phrase
dialog is forced at startup time.

**DH parameter interoperability with primes > 1024 bit**

Beginning with version 2.4.7, mod_ssl makes use of
standardized DH parameters with prime lengths of 2048, 3072
and 4096 bits and with additional prime lengths of 6144 and
8192 bits beginning with version 2.4.10 (from RFC 3526), and
hands them out to clients based on the length of the certificate's
RSA/DSA key. With Java-based clients in particular (Java 7 or
earlier), this may lead to handshake failures - see this FAQ
answer for working around such issues.

**Default DH parameters when using multiple certificates and
OpenSSL versions prior to 1.0.2**

When using multiple certificates to support different
authentication algorithms (like RSA, DSA, but mainly ECC) and
OpenSSL prior to 1.0.2, it is recommended to either use custom
DH parameters (preferably) by adding them to the first certificate
file (as described above), or to order the
$\texttt{SSLCertificateFile}$ directives such that RSA/DSA
certificates are placed after the ECC one.

This is due to a limitation in older versions of OpenSSL which
don't let the Apache HTTP Server determine the currently
selected certificate at handshake time (when the DH parameters
must be sent to the peer) but instead always provide the last
configured certificate. Consequently, the server may select
default DH parameters based on the length of the wrong
certificate's key (ECC keys are much smaller than RSA/DSA ones and their length is not relevant for selecting DH primes).

Since custom DH parameters always take precedence over the default ones, this issue can be avoided by creating and configuring them (as described above), thus using a custom/suitable length.

Example

SSLCertificateFile "/usr/local/apache2/conf/ssl.crt/server.crt"
**Description:** Server PEM-encoded private key file

**Syntax:**

```plaintext
SSLCertificateKeyFile file-path
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive points to the PEM-encoded private key file for the server. If the contained private key is encrypted, the pass phrase dialog is forced at startup time.

The directive can be used multiple times (referencing different filenames) to support multiple algorithms for server authentication. For each `SSLCertificateKeyFile` directive, there must be a matching `SSLCertificateFile` directive.

The private key may also be combined with the certificate in the file given by `SSLCertificateFile`, but this practice is highly discouraged. If it is used, the certificate files using such an embedded key must be configured after the certificates using a separate key file.

**Example**

```plaintext
SSLCertificateKeyFile "/usr/local/apache2/conf/ssl.key/server.key"
```
**Description:** Cipher Suite available for negotiation in SSL handshake

**Syntax:** SSLCipherSuite cipher-spec

**Default:** SSLCipherSuite DEFAULT (depends on OpenSSL version)

**Context:** server config, virtual host, directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_ssl

This complex directive uses a colon-separated `cipher-spec` string consisting of OpenSSL cipher specifications to configure the Cipher Suite the client is permitted to negotiate in the SSL handshake phase. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured Cipher Suite after the HTTP request was read but before the HTTP response is sent.

An SSL cipher specification in `cipher-spec` is composed of 4 major attributes plus a few extra minor ones:

- **Key Exchange Algorithm:**
  RSA, Diffie-Hellman, Elliptic Curve Diffie-Hellman, Secure Remote Password
- **Authentication Algorithm:**
  RSA, Diffie-Hellman, DSS, ECDSA, or none.
- **Cipher/Encryption Algorithm:**
  AES, DES, Triple-DES, RC4, RC2, IDEA, etc.
- **MAC Digest Algorithm:**
  MD5, SHA or SHA1, SHA256, SHA384.
An SSL cipher can also be an export cipher. SSLv2 ciphers are no longer supported. To specify which ciphers to use, one can either specify all the Ciphers, one at a time, or use aliases to specify the preference and order for the ciphers (see Table 1). The actually available ciphers and aliases depends on the used openssl version. Newer openssl versions may include additional ciphers.

<table>
<thead>
<tr>
<th>Tag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Exchange Algorithm:</td>
<td></td>
</tr>
<tr>
<td>kRSA</td>
<td>RSA key exchange</td>
</tr>
<tr>
<td>kDHr</td>
<td>Diffie-Hellman key exchange with RSA key</td>
</tr>
<tr>
<td>kDHd</td>
<td>Diffie-Hellman key exchange with DSA key</td>
</tr>
<tr>
<td>kEDH</td>
<td>Ephemeral (temp.key) Diffie-Hellman key exchange (no cert)</td>
</tr>
<tr>
<td>kSRP</td>
<td>Secure Remote Password (SRP) key exchange</td>
</tr>
<tr>
<td>Authentication Algorithm:</td>
<td></td>
</tr>
<tr>
<td>aNULL</td>
<td>No authentication</td>
</tr>
<tr>
<td>aRSA</td>
<td>RSA authentication</td>
</tr>
<tr>
<td>aDSS</td>
<td>DSS authentication</td>
</tr>
<tr>
<td>aDH</td>
<td>Diffie-Hellman authentication</td>
</tr>
<tr>
<td>Cipher Encoding Algorithm:</td>
<td></td>
</tr>
<tr>
<td>eNULL</td>
<td>No encryption</td>
</tr>
<tr>
<td>NULL</td>
<td>alias for eNULL</td>
</tr>
<tr>
<td>AES</td>
<td>AES encryption</td>
</tr>
<tr>
<td>DES</td>
<td>DES encryption</td>
</tr>
<tr>
<td>3DES</td>
<td>Triple-DES encryption</td>
</tr>
<tr>
<td>RC4</td>
<td>RC4 encryption</td>
</tr>
<tr>
<td>RC2</td>
<td>RC2 encryption</td>
</tr>
<tr>
<td>IDEA</td>
<td>IDEA encryption</td>
</tr>
<tr>
<td>MAC Digest Algorithm:</td>
<td></td>
</tr>
<tr>
<td>MD5</td>
<td>MD5 hash function</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------</td>
</tr>
<tr>
<td>SHA1</td>
<td>SHA1 hash function</td>
</tr>
<tr>
<td>SHA</td>
<td>alias for SHA1</td>
</tr>
<tr>
<td>SHA256</td>
<td>SHA256 hash function</td>
</tr>
<tr>
<td>SHA384</td>
<td>SHA384 hash function</td>
</tr>
<tr>
<td><strong>Aliases:</strong></td>
<td></td>
</tr>
<tr>
<td>SSLv3</td>
<td>all SSL version 3.0 ciphers</td>
</tr>
<tr>
<td>TLSv1</td>
<td>all TLS version 1.0 ciphers</td>
</tr>
<tr>
<td>EXP</td>
<td>all export ciphers</td>
</tr>
<tr>
<td>EXPORT40</td>
<td>all 40-bit export ciphers only</td>
</tr>
<tr>
<td>EXPORT56</td>
<td>all 56-bit export ciphers only</td>
</tr>
<tr>
<td>LOW</td>
<td>all low strength ciphers (no export, single DES)</td>
</tr>
<tr>
<td>MEDIUM</td>
<td>all ciphers with 128 bit encryption</td>
</tr>
<tr>
<td>HIGH</td>
<td>all ciphers using Triple-DES</td>
</tr>
<tr>
<td>RSA</td>
<td>all ciphers using RSA key exchange</td>
</tr>
<tr>
<td>DH</td>
<td>all ciphers using Diffie-Hellman key exchange</td>
</tr>
<tr>
<td>EDH</td>
<td>all ciphers using Ephemeral Diffie-Hellman key exchange</td>
</tr>
<tr>
<td>ECDH</td>
<td>Elliptic Curve Diffie-Hellman key exchange</td>
</tr>
<tr>
<td>ADH</td>
<td>all ciphers using Anonymous Diffie-Hellman key exchange</td>
</tr>
<tr>
<td>AECRH</td>
<td>all ciphers using Anonymous Elliptic Curve Diffie-Hellman key exchange</td>
</tr>
<tr>
<td>SRP</td>
<td>all ciphers using Secure Remote Password (SRP) key exchange</td>
</tr>
<tr>
<td>DSS</td>
<td>all ciphers using DSS authentication</td>
</tr>
<tr>
<td>ECDSA</td>
<td>all ciphers using ECDSA authentication</td>
</tr>
<tr>
<td>aNULL</td>
<td>all ciphers using no authentication</td>
</tr>
</tbody>
</table>

Now where this becomes interesting is that these can be put
together to specify the order and ciphers you wish to use. To speed this up there are also aliases (SSLv3, TLSv1, EXP, LOW, MEDIUM, HIGH) for certain groups of ciphers. These tags can be joined together with prefixes to form the cipher-spec. Available prefixes are:

- none: add cipher to list
- +: move matching ciphers to the current location in list
- -: remove cipher from list (can be added later again)
- !: kill cipher from list completely (can not be added later again)

**aNULL, eNULL and EXP ciphers are always disabled**

Beginning with version 2.4.7, null and export-grade ciphers are always disabled, as mod_ssl unconditionally adds !aNULL!: !eNULL!: !EXP to any cipher string at initialization.

A simpler way to look at all of this is to use the `openssl ciphers -v` command which provides a nice way to successively create the correct cipher-spec string. The default cipher-spec string depends on the version of the OpenSSL libraries used. Let's suppose it is `RC4-SHA:AES128-SHA:HIGH:MEDIUM:!aNULL!:!MD5` which means the following: Put RC4-SHA and AES128-SHA at the beginning. We do this, because these ciphers offer a good compromise between speed and security. Next, include high and medium security ciphers. Finally, remove all ciphers which do not authenticate, i.e. for SSL the Anonymous Diffie-Hellman ciphers, as well as all ciphers which use MD5 as hash algorithm, because it has been proven insufficient.

```
$ openssl ciphers -v 'RC4-SHA:AES128-SHA:HIGH:MEDIUM:!aNULL!:MD5'
RC4-SHA  SSLv3  Kx=RSA  Au=RSA  Enc=RC4(128)  MAC=MD5
AES128-SHA SSLv3  Kx=RSA  Au=RSA  Enc=AES(128)  MAC=MD5
```
DHE-RSA-AES256-SHA | SSLv3 | Kx=DH | Au=RSA | Enc=AES(256) | Mac=SHA1
--- | --- | --- | --- | --- | ---
SEED-SHA | SSLv3 | Kx=RSA | Au=RSA | Enc=SEED(128) | Mac=SHA1
PSK-RC4-SHA | SSLv3 | Kx=PSK | Au=PSK | Enc=RC4(128) | Mac=SHA1
KRB5-RC4-SHA | SSLv3 | Kx=KRB5 | Au=KRB5 | Enc=RC4(128) | Mac=SHA1

The complete list of particular RSA & DH ciphers for SSL is given in Table 2.

**Example**

SSLCipherSuite RSA:!EXP:!NULL:+HIGH:+MEDIUM:-LOW

<table>
<thead>
<tr>
<th>Cipher-Tag</th>
<th>Protocol</th>
<th>Key Ex.</th>
<th>Auth.</th>
<th>Enc.</th>
<th>MAC</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RSA Ciphers:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DES-CBC3-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>3DES(168)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>IDEA-CBC-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>IDEA(128)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>RC4-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>RC4(128)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>RC4-MD5</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>RC4(128)</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>DES-CBC-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>DES(56)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EXP-DES-CBC-SHA</td>
<td>SSLv3</td>
<td>RSA(512)</td>
<td>RSA</td>
<td>DES(40)</td>
<td>SHA1</td>
<td>export</td>
</tr>
<tr>
<td>EXP-RC2-CBC-MD5</td>
<td>SSLv3</td>
<td>RSA(512)</td>
<td>RSA</td>
<td>RC2(40)</td>
<td>MD5</td>
<td>export</td>
</tr>
<tr>
<td>EXP-RC4-MD5</td>
<td>SSLv3</td>
<td>RSA(512)</td>
<td>RSA</td>
<td>RC4(40)</td>
<td>MD5</td>
<td>export</td>
</tr>
<tr>
<td>NULL-SHA</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>None</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>NULL-MD5</td>
<td>SSLv3</td>
<td>RSA</td>
<td>RSA</td>
<td>None</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td><strong>Diffie-Hellman Ciphers:</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADH-DES-</td>
<td>SSLv3</td>
<td>DH</td>
<td>None</td>
<td>3DES(168)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>CBC3 - SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>None</td>
<td>DES(56)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>-------</td>
<td>----</td>
<td>------</td>
<td>---------</td>
<td>------</td>
<td></td>
</tr>
<tr>
<td>ADH - DES - CBC - SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>None</td>
<td>DES(56)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>ADH - RC4 - MD5</td>
<td>SSLv3</td>
<td>DH</td>
<td>None</td>
<td>RC4(128)</td>
<td>MD5</td>
<td></td>
</tr>
<tr>
<td>EDH - RSA - DES - CBC3 - SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>RSA</td>
<td>3DES(168)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EDH - DSS - DES - CBC3 - SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>DSS</td>
<td>3DES(168)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EDH - RSA - DES - CBC - SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>RSA</td>
<td>DES(56)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EDH - DSS - DES - CBC - SHA</td>
<td>SSLv3</td>
<td>DH</td>
<td>DSS</td>
<td>DES(56)</td>
<td>SHA1</td>
<td></td>
</tr>
<tr>
<td>EXP - EDH - RSA - DES - CBC - SHA</td>
<td>SSLv3</td>
<td>DH(512)</td>
<td>RSA</td>
<td>DES(40)</td>
<td>SHA1 export</td>
<td></td>
</tr>
<tr>
<td>EXP - EDH - DSS - DES - CBC - SHA</td>
<td>SSLv3</td>
<td>DH(512)</td>
<td>DSS</td>
<td>DES(40)</td>
<td>SHA1 export</td>
<td></td>
</tr>
<tr>
<td>EXP - ADH - DES - CBC - SHA</td>
<td>SSLv3</td>
<td>DH(512)</td>
<td>None</td>
<td>DES(40)</td>
<td>SHA1 export</td>
<td></td>
</tr>
<tr>
<td>EXP - ADH - RC4 - MD5</td>
<td>SSLv3</td>
<td>DH(512)</td>
<td>None</td>
<td>RC4(40)</td>
<td>MD5 export</td>
<td></td>
</tr>
</tbody>
</table>
**Description:** Enable compression on the SSL level

**Syntax:** SSLCompression on|off

**Default:** SSLCompression off

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available in httpd 2.4.3 and later, if using OpenSSL 0.9.8 or later; virtual host scope available if using OpenSSL 1.0.0 or later. The default used to be on in version 2.4.3.

This directive allows to enable compression on the SSL level.

Enabling compression causes security issues in most setups (the so called CRIME attack).
**SSLCryptoDevice Directive**

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Enable use of a cryptographic hardware accelerator</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td><code>SSLCryptoDevice engine</code></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td><code>SSLCryptoDevice builtin</code></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ssl</td>
</tr>
</tbody>
</table>

This directive enables use of a cryptographic hardware accelerator board to offload some of the SSL processing overhead. This directive can only be used if the SSL toolkit is built with "engine" support; OpenSSL 0.9.7 and later releases have "engine" support by default, the separate "-engine" releases of OpenSSL 0.9.6 must be used.

To discover which engine names are supported, run the command "openssl engine".

**Example**

```
# For a Broadcom accelerator:
SSLCryptoDevice ubsec
```
This directive toggles the usage of the SSL/TLS Protocol Engine. This should be used inside a `<VirtualHost>` section to enable SSL/TLS for a virtual host. By default the SSL/TLS Protocol Engine is disabled for both the main server and all configured virtual hosts.

**Example**

```xml
<VirtualHost _default_:443>
  SSLEngine on
  #...
</VirtualHost>
```

In Apache 2.1 and later, `SSLEngine` can be set to `optional`. This enables support for [RFC 2817](https://tools.ietf.org/html/rfc2817), Upgrading to TLS Within HTTP/1.1. At this time no web browsers support RFC 2817.
**Description:** SSL FIPS mode Switch

**Syntax:** \texttt{SSLFIPS\ on|off}

**Default:** \texttt{SSLFIPS\ off}

**Context:** server config

**Status:** Extension

**Module:** mod_ssl

This directive toggles the usage of the SSL library FIPS\_mode flag. It must be set in the global server context and cannot be configured with conflicting settings (SSLFIPS\ on followed by SSLFIPS\ off or similar). The mode applies to all SSL library operations.

If httpd was compiled against an SSL library which did not support the FIPS\_mode flag, SSLFIPS\ on will fail. Refer to the FIPS 140-2 Security Policy document of the SSL provider library for specific requirements to use mod\_ssl in a FIPS 140-2 approved mode of operation; note that mod\_ssl itself is not validated, but may be described as using FIPS 140-2 validated cryptographic module, when all components are assembled and operated under the guidelines imposed by the applicable Security Policy.
**Description:** Option to prefer the server's cipher preference order

**Syntax:** SSLHonorCipherOrder on|off

**Default:** SSLHonorCipherOrder off

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

When choosing a cipher during an SSLv3 or TLSv1 handshake, normally the client's preference is used. If this directive is enabled, the server's preference will be used instead.

**Example**

SSLHonorCipherOrder on
**Description:** Option to enable support for insecure renegotiation

**Syntax:** SSLInsecureRenegotiation on|off

**Default:** SSLInsecureRenegotiation off

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available in httpd 2.2.15 and later, if using OpenSSL 0.9.8m or later

As originally specified, all versions of the SSL and TLS protocols (up to and including TLS/1.2) were vulnerable to a Man-in-the-Middle attack ([CVE-2009-3555](https://cve.mitre.org/cgi-bin/cvename.cgi?eid=CVE-2009-3555)) during a renegotiation. This vulnerability allowed an attacker to "prefix" a chosen plaintext to the HTTP request as seen by the web server. A protocol extension was developed which fixed this vulnerability if supported by both client and server.

If `mod_ssl` is linked against OpenSSL version 0.9.8m or later, by default renegotiation is only supported with clients supporting the new protocol extension. If this directive is enabled, renegotiation will be allowed with old (unpatched) clients, albeit insecurely.

**Security warning**

If this directive is enabled, SSL connections will be vulnerable to the Man-in-the-Middle prefix attack as described in [CVE-2009-3555](https://cve.mitre.org/cgi-bin/cvename.cgi?eid=CVE-2009-3555).

**Example**

SSLInsecureRenegotiation on
The SSL_SECURE_RENEG environment variable can be used from an SSI or CGI script to determine whether secure renegotiation is supported for a given SSL connection.
<table>
<thead>
<tr>
<th><strong>Description</strong></th>
<th>Set the default responder URI for OCSP validation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax</strong></td>
<td>SSLOCSDefaultResponder uri</td>
</tr>
<tr>
<td><strong>Context</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module</strong></td>
<td>mod_ssl</td>
</tr>
</tbody>
</table>

This option sets the default OCSP responder to use. If `SSLOCSP0verrudeResponder` is not enabled, the URI given will be used only if no responder URI is specified in the certificate being verified.
**Description:** Enable OCSP validation of the client certificate chain

**Syntax:** SSLOCSPEnable on|off

**Default:** SSLOCSPEnable off

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This option enables OCSP validation of the client certificate chain. If this option is enabled, certificates in the client's certificate chain will be validated against an OCSP responder after normal verification (including CRL checks) have taken place.

The OCSP responder used is either extracted from the certificate itself, or derived by configuration; see the SSLOCSPDefaultResponder and SSLOCSPOverrideResponder directives.

**Example**

SSLVerifyClient on
SSLOCSPEnable on
SSLOCSPDefaultResponder "http://responder.example.com:8888/responder"
SSLOCSPOverrideResponder on
| **Description:** | skip the OCSP responder certificates verification |
| **Syntax:** | SSL0CSPNoverify *On/Off* |
| **Default:** | SSL0CSPNoverify *Off* |
| **Context:** | server config, virtual host |
| **Status:** | Extension |
| **Module:** | mod_ssl |
| **Compatibility:** | Available in httpd 2.4.26 and later, if using OpenSSL 0.9.7 or later |

Skip the OCSP responder certificates verification, mostly useful when testing an OCSP server.
<table>
<thead>
<tr>
<th>Description</th>
<th>Force use of the default responder URI for OCSP validation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Syntax</td>
<td>SSLOCSPOverrideResponder on</td>
</tr>
<tr>
<td>Default</td>
<td>SSLOCSPOverrideResponder off</td>
</tr>
<tr>
<td>Context</td>
<td>server config, virtual host</td>
</tr>
<tr>
<td>Status</td>
<td>Extension</td>
</tr>
<tr>
<td>Module</td>
<td>mod_ssl</td>
</tr>
</tbody>
</table>

This option forces the configured default OCSP responder to be used during OCSP certificate validation, regardless of whether the certificate being validated references an OCSP responder.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Proxy URL to use for OCSP requests</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SSL0CSPProxYURL  url</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ssl</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in httpd 2.4.19 and later</td>
</tr>
</tbody>
</table>

This option allows to set the URL of a HTTP proxy that should be used for all queries to OCSP responders.
This supplies a list of trusted OCSP responder certificates to be used during OCSP responder certificate validation. The supplied certificates are implicitly trusted without any further validation. This is typically used where the OCSP responder certificate is self signed or omitted from the OCSP response.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Timeout for OCSP queries</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SSLOCSPResponderTimeout  seconds</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>SSLOCSPResponderTimeout  10</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ssl</td>
</tr>
</tbody>
</table>

This option sets the timeout for queries to OCSP responders, when SSLOCSPEnable is turned on.
**Description:** Maximum allowable age for OCSP responses

**Syntax:** SSL0CSPResponseMaxAge seconds

**Default:** SSL0CSPResponseMaxAge -1

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This option sets the maximum allowable age ("freshness") for OCSP responses. The default value (-1) does not enforce a maximum age, which means that OCSP responses are considered valid as long as their nextUpdate field is in the future.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Maximum allowable time skew for OCSP response validation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SSLOCSPResponseTimeSkew  seconds</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>SSLOCSPResponseTimeSkew  300</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ssl</td>
</tr>
</tbody>
</table>

This option sets the maximum allowable time skew for OCSP responses (when checking their thisUpdate and nextUpdate fields).
This option determines whether queries to OCSP responders should contain a nonce or not. By default, a query nonce is always used and checked against the response's one. When the responder does not use nonces (e.g. Microsoft OCSP Responder), this option should be turned off.
**Description:** Configure OpenSSL parameters through its SSL_CONF API

**Syntax:**
SSLOpenSSLConfCmd *command-name* *command-value*

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available in httpd 2.4.8 and later, if using OpenSSL 1.0.2 or later

This directive exposes OpenSSL's SSL_CONF API to mod_ssl, allowing a flexible configuration of OpenSSL parameters without the need of implementing additional mod_ssl directives when new features are added to OpenSSL.

The set of available SSLOpenSSLConfCmd commands depends on the OpenSSL version being used for mod_ssl (at least version 1.0.2 is required). For a list of supported command names, see the section Supported configuration file commands in the SSL_CONF_cmd(3) manual page for OpenSSL.

Some of the SSLOpenSSLConfCmd commands can be used as an alternative to existing directives (such as SSLCipherSuite or SSLProtocol), though it should be noted that the syntax / allowable values for the parameters may sometimes differ.

**Examples**

SSLOpenSSLConfCmd Options -SessionTicket,ServerPreference
SSLOpenSSLConfCmd ECDHParameters brainpoolP256r1
SSLOpenSSLConfCmd ServerInfoFile "/usr/local/apache2/conf/server-info.pem"
SSLOpenSSLConfCmd Protocol "-ALL, TLSv1.2"
SSLOpenSSLConfCmd SignatureAlgorithms RSA+SHA384:ECDSA+SHA256
**Description:** Configure various SSL engine run-time options

**Syntax:**

SSLOptions [+-]option ...

**Context:**

server config, virtual host, directory, .htaccess

**Override:**

Options

**Status:**

Extension

**Module:**

mod_ssl

This directive can be used to control various run-time options on a per-directory basis. Normally, if multiple SSLOptions could apply to a directory, then the most specific one is taken completely; the options are not merged. However if all the options on the SSLOptions directive are preceded by a plus (+) or minus (-) symbol, the options are merged. Any options preceded by a + are added to the options currently in force, and any options preceded by a - are removed from the options currently in force.

The available *options* are:

- **StdEnvVars**
  When this option is enabled, the standard set of SSL related CGI/SSI environment variables are created. This per default is disabled for performance reasons, because the information extraction step is a rather expensive operation. So one usually enables this option for CGI and SSI requests only.

- **ExportCertData**
  When this option is enabled, additional CGI/SSI environment variables are created: SSL_SERVER_CERT, SSL_CLIENT_CERT and SSL_CLIENT_CERT_CHAIN_n (with n = 0,1,2,..). These contain the PEM-encoded X.509 Certificates of server and client for the current HTTPS connection and can be used by CGI scripts for deeper Certificate checking. Additionally all other certificates of the
client certificate chain are provided, too. This bloats up the environment a little bit which is why you have to use this option to enable it on demand.

- **FakeBasicAuth**
  When this option is enabled, the Subject Distinguished Name (DN) of the Client X509 Certificate is translated into a HTTP Basic Authorization username. This means that the standard Apache authentication methods can be used for access control. The user name is just the Subject of the Client's X509 Certificate (can be determined by running OpenSSL's `openssl x509` command: `openssl x509 -noout -subject -in certificate.crt`). Note that no password is obtained from the user. Every entry in the user file needs this password: ``xxj31ZMTZzkVA``, which is the DES-encrypted version of the word `password`. Those who live under MD5-based encryption (for instance under FreeBSD or BSD/OS, etc.) should use the following MD5 hash of the same word: ``$1$OXLyS...$Owx8s2/m9/gkcRvXzgoE/``.

  Note that the [AuthBasicFake](#) directive within the [mod_auth_basic](#) can be used as a more general mechanism for faking basic authentication, giving control over the structure of both the username and password.

- **StrictRequire**
  This forces forbidden access when SSLRequireSSL or SSLRequire successfully decided that access should be forbidden. Usually the default is that in the case where a `Satisfy any` directive is used, and other access restrictions are passed, denial of access due to SSLRequireSSL or SSLRequire is overridden (because that's how the Apache Satisfy mechanism should work.) But for strict access restriction you can use SSLRequireSSL
and/or SSLRequire in combination with an `SSLOptions +StrictRequire`.
Then an additional `Satisfy Any` has no chance once mod_ssl has decided to deny access.

- **OptRenegotiate**
  This enables optimized SSL connection renegotiation handling when SSL directives are used in per-directory context. By default a strict scheme is enabled where every per-directory reconfiguration of SSL parameters causes a full SSL renegotiation handshake. When this option is used mod_ssl tries to avoid unnecessary handshakes by doing more granular (but still safe) parameter checks. Nevertheless these granular checks sometimes may not be what the user expects, so enable this on a per-directory basis only, please.

- **LegacyDNStringFormat**
  This option influences how values of the SSL_{CLIENT,SERVER}_{I,S}_DN variables are formatted. Since version 2.3.11, Apache HTTPD uses a RFC 2253 compatible format by default. This uses commas as delimiters between the attributes, allows the use of non-ASCII characters (which are converted to UTF8), escapes various special characters with backslashes, and sorts the attributes with the "C" attribute last.

  If LegacyDNStringFormat is set, the old format will be used which sorts the "C" attribute first, uses slashes as separators, and does not handle non-ASCII and special characters in any consistent way.

**Example**

```plaintext
SSLOptions +FakeBasicAuth -StrictRequire
<Files ~ "\.(cgi|shtml)$">
  SSLOptions +StdEnvVars -ExportCertData
</Files>
```
**Description:** Type of pass phrase dialog for encrypted private keys

**Syntax:** `SSLPassPhraseDialog type`

**Default:** `SSLPassPhraseDialog builtin`

**Context:** `serverconfig`

**Status:** `Extension`

**Module:** `mod_ssl`

When Apache starts up it has to read the various Certificate (see `SSLCertificateFile`) and Private Key (see `SSLCertificateKeyFile`) files of the SSL-enabled virtual servers. Because for security reasons the Private Key files are usually encrypted, `mod_ssl` needs to query the administrator for a Pass Phrase in order to decrypt those files. This query can be done in two ways which can be configured by `type`:

- **builtin**
  This is the default where an interactive terminal dialog occurs at startup time just before Apache detaches from the terminal. Here the administrator has to manually enter the Pass Phrase for each encrypted Private Key file. Because a lot of SSL-enabled virtual hosts can be configured, the following reuse-scheme is used to minimize the dialog: When a Private Key file is encrypted, all known Pass Phrases (at the beginning there are none, of course) are tried. If one of those known Pass Phrases succeeds no dialog pops up for this particular Private Key file. If none succeeded, another Pass Phrase is queried on the terminal and remembered for the next round (where it perhaps can be reused).

This scheme allows `mod_ssl` to be maximally flexible (because for N encrypted Private Key files you can use N different Pass Phrases - but then you have to enter all of
them, of course) while minimizing the terminal dialog (i.e. when you use a single Pass Phrase for all N Private Key files this Pass Phrase is queried only once).

- `/path/to/program [args...]`
  This mode allows an external program to be used which acts as a pipe to a particular input device; the program is sent the standard prompt text used for the builtin mode on stdin, and is expected to write password strings on stdout. If several passwords are needed (or an incorrect password is entered), additional prompt text will be written subsequent to the first password being returned, and more passwords must then be written back.

- `exec:/path/to/program`
  Here an external program is configured which is called at startup for each encrypted Private Key file. It is called with two arguments (the first is of the form `servername:portnumber`, the second is either `RSA`, `DSA`, `ECC` or an integer index starting at 3 if more than three keys are configured), which indicate for which server and algorithm it has to print the corresponding Pass Phrase to stdout. In versions 2.4.8 (unreleased) and 2.4.9, it is called with one argument, a string of the form `servername:portnumber:index` (with index being a zero-based integer number), which indicate the server, TCP port and certificate number. The intent is that this external program first runs security checks to make sure that the system is not compromised by an attacker, and only when these checks were passed successfully it provides the Pass Phrase.

Both these security checks, and the way the Pass Phrase is determined, can be as complex as you like. Mod_ssl just
defines the interface: an executable program which provides the Pass Phrase on stdout. Nothing more or less! So, if you're really paranoid about security, here is your interface. Anything else has to be left as an exercise to the administrator, because local security requirements are so different.

The reuse-algorithm above is used here, too. In other words: The external program is called only once per unique Pass Phrase.

```
Example
SSLPassPhraseDialog "exec:/usr/local/apache/sbin/pp-filter"
```
**Description:** Configure usable SSL/TLS protocol versions

**Syntax:**
SSLProtocol [+|+]protocol ...

**Default:**
SSLProtocol all -SSLv3 (up to 2.4.16: all)

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive can be used to control which versions of the SSL/TLS protocol will be accepted in new connections.

The available (case-insensitive) **protocols** are:

- **SSLv3**
  This is the Secure Sockets Layer (SSL) protocol, version 3.0, from the Netscape Corporation. It is the successor to SSLv2 and the predecessor to TLSv1, but is deprecated in [RFC 7568](https://tools.ietf.org/html/rfc7568).

- **TLSv1**
  This is the Transport Layer Security (TLS) protocol, version 1.0. It is the successor to SSLv3 and is defined in [RFC 2246](https://tools.ietf.org/html/rfc2246). It is supported by nearly every client.

- **TLSv1.1** (when using OpenSSL 1.0.1 and later)
  A revision of the TLS 1.0 protocol, as defined in [RFC 4346](https://tools.ietf.org/html/rfc4346).

- **TLSv1.2** (when using OpenSSL 1.0.1 and later)
  A revision of the TLS 1.1 protocol, as defined in [RFC 5246](https://tools.ietf.org/html/rfc5246).

- **all**
  This is a shortcut for ``+SSLv3 +TLSv1`` or - when using OpenSSL 1.0.1 and later - ``+SSLv3 +TLSv1 +TLSv1.1``
+TLSv1.2", respectively (except for OpenSSL versions compiled with the "no-ssl3" configuration option, where all does not include +SSLv3).

**Example**

SSLProtocol TLSv1
**Description:** File of concatenated PEM-encoded CA Certificates for Remote Server Auth

**Syntax:**
```
SSLProxyCACertificateFile file-path
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificates of Certification Authorities (CA) whose *remote servers* you deal with. These are used for Remote Server Authentication. Such a file is simply the concatenation of the various PEM-encoded Certificate files, in order of preference. This can be used alternatively and/or additionally to `SSLProxyCACertificatePath`.

**Example**
```
SSLProxyCACertificateFile "/usr/local/apache2/conf/ssl.crt/ca-bundle-remote-server.crt"
```
**Description:** Directory of PEM-encoded CA Certificates for Remote Server Auth

**Syntax:** SSLProxyCACertificatePath directory-path

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive sets the directory where you keep the Certificates of Certification Authorities (CAs) whose remote servers you deal with. These are used to verify the remote server certificate on Remote Server Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you can't just place the Certificate files there: you also have to create symbolic links named hash-value . N. And you should always make sure this directory contains the appropriate symbolic links.

**Example**

SSLProxyCACertificatePath "/usr/local/apache2/conf/ssl.crt/"
### SSLProxyCARevocationCheck

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Enable CRL-based revocation checking for Remote Server Auth</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SSLProxyCARevocationCheck chain</td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>SSLProxyCARevocationCheck none</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ssl</td>
</tr>
</tbody>
</table>

Enables certificate revocation list (CRL) checking for the *remote servers* you deal with. At least one of `SSLProxyCARevocationFile` or `SSLProxyCARevocationPath` must be configured. When set to `chain` (recommended setting), CRL checks are applied to all certificates in the chain, while setting it to `leaf` limits the checks to the end-entity cert.

**When set to chain or leaf, CRLs must be available for successful validation**

Prior to version 2.3.15, CRL checking in mod_ssl also succeeded when no CRL(s) were found in any of the locations configured with `SSLProxyCARevocationFile` or `SSLProxyCARevocationPath`. With the introduction of this directive, the behavior has been changed: when checking is enabled, CRLs *must* be present for the validation to succeed - otherwise it will fail with an "unable to get certificate CRL" error.

**Example**

SSLProxyCARevocationCheck chain
**Description:** File of concatenated PEM-encoded CA CRLs for Remote Server Auth

**Syntax:** `SSLProxyCARevocationFile file-path`

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive sets the *all-in-one* file where you can assemble the Certificate Revocation Lists (CRL) of Certification Authorities (CA) whose remote servers you deal with. These are used for Remote Server Authentication. Such a file is simply the concatenation of the various PEM-encoded CRL files, in order of preference. This can be used alternatively and/or additionally to `SSLProxyCARevocationPath`.

**Example**

```
SSLProxyCARevocationFile "/usr/local/apache2/conf/ssl.crl/ca-bundle-remote-server.crl"
```

```
**Description:** Directory of PEM-encoded CA CRLs for Remote Server Auth

**Syntax:**

```
SSLProxyCARevocationPath directory-path
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive sets the directory where you keep the Certificate Revocation Lists (CRL) of Certification Authorities (CAs) whose remote servers you deal with. These are used to revoke the remote server certificate on Remote Server Authentication.

The files in this directory have to be PEM-encoded and are accessed through hash filenames. So usually you have not only to place the CRL files there. Additionally you have to create symbolic links named `hash-value`.rN. And you should always make sure this directory contains the appropriate symbolic links.

**Example**

```
SSLProxyCARevocationPath "/usr/local/apache2/conf/ssl.crl/"
```
Description:  Whether to check the remote server certificate's CN field

Syntax:  SSLProxyCheckPeerCN on|off

Default:  SSLProxyCheckPeerCN on

Context:  server config, virtual host

Status:  Extension

Module:  mod_ssl

This directive sets whether the remote server certificate's CN field is compared against the hostname of the request URL. If both are not equal a 502 status code (Bad Gateway) is sent. SSLProxyCheckPeerCN is superseded by SSLProxyCheckPeerName in release 2.4.5 and later.

In all releases 2.4.5 through 2.4.20, setting SSLProxyCheckPeerName off was sufficient to enable this behavior (as the SSLProxyCheckPeerCN default was on.) In these releases, both directives must be set to off to completely avoid remote server certificate name validation. Many users reported this to be very confusing.

As of release 2.4.21, all configurations which enable either one of the SSLProxyCheckPeerName or SSLProxyCheckPeerCN options will use the new SSLProxyCheckPeerName behavior, and all configurations which disable either one of the SSLProxyCheckPeerName or SSLProxyCheckPeerCN options will suppress all remote server certificate name validation. Only the following configuration will trigger the legacy certificate CN comparison in 2.4.21 and later releases;

Example
SSLProxyCheckPeerCN on
SSLProxyCheckPeerName off
**Description:** Whether to check if remote server certificate is expired

**Syntax:** SSLProxyCheckPeerExpire on|off

**Default:** SSLProxyCheckPeerExpire on

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive sets whether it is checked if the remote server certificate is expired or not. If the check fails a 502 status code (Bad Gateway) is sent.

**Example**

SSLProxyCheckPeerExpire on
**Description:** Configure host name checking for remote server certificates

**Syntax:** SSLProxyCheckPeerName on|off

**Default:** SSLProxyCheckPeerName on

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Apache HTTP Server 2.4.5 and later

This directive configures host name checking for server certificates when mod_ssl is acting as an SSL client. The check will succeed if the host name from the request URI matches one of the CN attribute(s) of the certificate's subject, or matches the subjectAltName extension. If the check fails, the SSL request is aborted and a 502 status code (Bad Gateway) is returned.

Wildcard matching is supported for specific cases: an subjectAltName entry of type dNSName, or CN attributes starting with *. will match with any host name of the same number of name elements and the same suffix. E.g. *.example.org will match foo.example.org, but will not match foo.bar.example.org, because the number of elements in the respective host names differs.

This feature was introduced in 2.4.5 and superseded the behavior of the SSLProxyCheckPeerCN directive, which only tested the exact value in the first CN attribute against the host name. However, many users were confused by the behavior of using these directives individually, so the mutual behavior of SSLProxyCheckPeerName and SSLProxyCheckPeerCN directives were improved in release 2.4.21. See the SSLProxyCheckPeerCN directive description for the original
behavior and details of these improvements.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Cipher Suite available for negotiation in SSL proxy handshake</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SSLProxyCipherSuite <em>cipher-spec</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>SSLProxyCipherSuite ALL:!ADH:RC4+RSA:+HIGH:+MEDIUM:+LOW:+E</td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host, directory, .htaccess</td>
</tr>
<tr>
<td><strong>Override:</strong></td>
<td>AuthConfig</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ssl</td>
</tr>
</tbody>
</table>

Equivalent to [SSLCipherSuite](#), but for the proxy connection. Please refer to [SSLCipherSuite](#) for additional information.
**Description:** SSL Proxy Engine Operation Switch

**Syntax:**

```
SSLProxyEngine on|off
```

**Default:**

```
SSLProxyEngine off
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive toggles the usage of the SSL/TLS Protocol Engine for proxy. This is usually used inside a `<VirtualHost>` section to enable SSL/TLS for proxy usage in a particular virtual host. By default the SSL/TLS Protocol Engine is disabled for proxy both for the main server and all configured virtual hosts.

Note that the `SSLProxyEngine` directive should not, in general, be included in a virtual host that will be acting as a forward proxy (using `<Proxy>` or `ProxyRequests` directives). `SSLProxyEngine` is not required to enable a forward proxy server to proxy SSL/TLS requests.

**Example**

```
<VirtualHost _default_:443>
  SSLProxyEngine on
  #...
</VirtualHost>
```
**Description:** File of concatenated PEM-encoded CA certificates to be used by the proxy for choosing a certificate.

**Syntax:**

```
SSLProxyMachineCertificateChainFile filename
```

**Context:** server config

**Override:** Not applicable

**Status:** Extension

**Module:** mod_ssl

---

This directive sets the all-in-one file where you keep the certificate chain for all of the client certs in use. This directive will be needed if the remote server presents a list of CA certificates that are not direct signers of one of the configured client certificates.

This referenced file is simply the concatenation of the various PEM-encoded certificate files. Upon startup, each client certificate configured will be examined and a chain of trust will be constructed.

---

**Security warning**

If this directive is enabled, all of the certificates in the file will be trusted as if they were also in **SSLProxyCACertificateFile**.

---

**Example**

```
```
**Description:** File of concatenated PEM-encoded client certificates and keys to be used by the proxy

**Syntax:**
```
SSLProxyMachineCertificateFile filename
```

**Context:** server config

**Override:** Not applicable

**Status:** Extension

**Module:** mod_ssl

This directive sets the all-in-one file where you keep the certificates and keys used for authentication of the proxy server to remote servers.

This referenced file is simply the concatenation of the various PEM-encoded certificate files, in order of preference. Use this directive alternatively or additionally to `SSLProxyMachineCertificatePath`.

Currently there is no support for encrypted private keys

**Example**
```
SSLProxyMachineCertificateFile "/usr/local/apache2/conf/ssl.crt,.......
```
**Description:** Directory of PEM-encoded client certificates and keys to be used by the proxy

**Syntax:**

```
SSLProxyMachineCertificatePath
directory
```

**Context:** server config

**Override:** Not applicable

**Status:** Extension

**Module:** mod_ssl

This directive sets the directory where you keep the certificates and keys used for authentication of the proxy server to remote servers.

The files in this directory must be PEM-encoded and are accessed through hash filenames. Additionally, you must create symbolic links named `hash-value`.N. And you should always make sure this directory contains the appropriate symbolic links.

Currently there is no support for encrypted private keys

**Example**

```
```
**Description:** Configure usable SSL protocol flavors for proxy usage

**Syntax:**
SSLProxyProtocol [+-]protocol ...

**Default:**
SSLProxyProtocol all -SSLv3 (up to 2.4.16: all)

**Context:** server config, virtual host

**Override:** Options

**Status:** Extension

**Module:** mod_ssl

This directive can be used to control the SSL protocol flavors mod_ssl should use when establishing its server environment for proxy. It will only connect to servers using one of the provided protocols.

Please refer to [SSLProtocol](#) for additional information.
**Description:** Type of remote server Certificate verification

**Syntax:**
SSLProxyVerify *level*

**Default:**
SSLProxyVerify none

**Context:**
server config, virtual host

**Status:**
Extension

**Module:**
mod_ssl

When a proxy is configured to forward requests to a remote SSL server, this directive can be used to configure certificate verification of the remote server.

The following levels are available for *level*:

- **none**: no remote server Certificate is required at all
- **optional**: the remote server *may* present a valid Certificate
- **require**: the remote server *has to* present a valid Certificate
- **optional_no_ca**: the remote server may present a valid Certificate but it need not to be (successfully) verifiable.

In practice only levels **none** and **require** are really interesting, because level **optional** doesn't work with all servers and level **optional_no_ca** is actually against the idea of authentication (but can be used to establish SSL test pages, etc.)

**Example**
SSLProxyVerify require
**Description:** Maximum depth of CA Certificates in Remote Server Certificate verification

**Syntax:** SSLProxyVerifyDepth *number*

**Default:** SSLProxyVerifyDepth 1

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

This directive sets how deeply mod_ssl should verify before deciding that the remote server does not have a valid certificate.

The depth actually is the maximum number of intermediate certificate issuers, i.e. the number of CA certificates which are max allowed to be followed while verifying the remote server certificate. A depth of 0 means that self-signed remote server certificates are accepted only, the default depth of 1 means the remote server certificate can be self-signed or has to be signed by a CA which is directly known to the server (i.e. the CA’s certificate is under SSLProxyCACertificatePath), etc.

**Example**

SSLProxyVerifyDepth 10
**Description:** Pseudo Random Number Generator (PRNG) seeding source

**Syntax:** `SSLRandomSeed context source [bytes]`

**Context:** server config

**Status:** Extension

**Module:** mod_ssl

This configures one or more sources for seeding the Pseudo Random Number Generator (PRNG) in OpenSSL at startup time (context is startup) and/or just before a new SSL connection is established (context is connect). This directive can only be used in the global server context because the PRNG is a global facility.

The following source variants are available:

- **builtin**
  This is the always available builtin seeding source. Its usage consumes minimum CPU cycles under runtime and hence can be always used without drawbacks. The source used for seeding the PRNG contains of the current time, the current process id and (when applicable) a randomly chosen 1KB extract of the inter-process scoreboard structure of Apache. The drawback is that this is not really a strong source and at startup time (where the scoreboard is still not available) this source just produces a few bytes of entropy. So you should always, at least for the startup, use an additional seeding source.

- **file:/path/to/source**
  This variant uses an external file /path/to/source as the source for seeding the PRNG. When *bytes* is specified, only the first *bytes* number of bytes of the file form the entropy (and *bytes* is given to /path/to/source as the first
argument). When \textit{bytes} is not specified the whole file forms the entropy (and 0 is given to \texttt{/path/to/source} as the first argument). Use this especially at startup time, for instance with an available \texttt{/dev/random} and/or \texttt{/dev/urandom} devices (which usually exist on modern Unix derivatives like FreeBSD and Linux).

\textbf{But be careful:} Usually \texttt{/dev/random} provides only as much entropy data as it actually has, i.e. when you request 512 bytes of entropy, but the device currently has only 100 bytes available two things can happen: On some platforms you receive only the 100 bytes while on other platforms the read blocks until enough bytes are available (which can take a long time). Here using an existing \texttt{/dev/urandom} is better, because it never blocks and actually gives the amount of requested data. The drawback is just that the quality of the received data may not be the best.

- \texttt{exec:/path/to/program}
  This variant uses an external executable \texttt{/path/to/program} as the source for seeding the PRNG. When \textit{bytes} is specified, only the first \textit{bytes} number of bytes of its \texttt{stdout} contents form the entropy. When \textit{bytes} is not specified, the entirety of the data produced on \texttt{stdout} form the entropy. Use this only at startup time when you need a very strong seeding with the help of an external program (for instance as in the example above with the \texttt{truerand} utility you can find in the mod_ssl distribution which is based on the AT&T \texttt{truerand} library). Using this in the connection context slows down the server too dramatically, of course. So usually you should avoid using external programs in that context.

- \texttt{egd:/path/to/egd-socket} (Unix only)
  This variant uses the Unix domain socket of the external
Entropy Gathering Daemon (EGD) (see http://www.lothar.com/tech/crypto/) to seed the PRNG. Use this if no random device exists on your platform.

**Example**

SSLRandomSeed startup builtin
SSLRandomSeed startup "file:/dev/random"
SSLRandomSeed startup "file:/dev/urandom" 1024
SSLRandomSeed startup "exec:/usr/local/bin/truerand" 16
SSLRandomSeed connect builtin
SSLRandomSeed connect "file:/dev/random"
SSLRandomSeed connect "file:/dev/urandom" 1024
**Description:** Set the size for the SSL renegotiation buffer

**Syntax:** SSLRenegBufferSize bytes

**Default:** SSLRenegBufferSize 131072

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_ssl

If an SSL renegotiation is required in per-location context, for example, any use of [SSLVerifyClient](#) in a Directory or Location block, then mod_ssl must buffer any HTTP request body into memory until the new SSL handshake can be performed. This directive can be used to set the amount of memory that will be used for this buffer.

Note that in many configurations, the client sending the request body will be untrusted so a denial of service attack by consumption of memory must be considered when changing this configuration setting.

**Example**

SSLRenegBufferSize 262144
**Description:** Allow access only when an arbitrarily complex boolean expression is true

**Syntax:** SSLRequire *expression*

**Context:** directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_ssl

---

**SSLRequire is deprecated**

SSLRequire is deprecated and should in general be replaced by `Require expr`. The so called `ap_expr` syntax of Require expr is a superset of the syntax of SSLRequire, with the following exception:

In SSLRequire, the comparison operators `<, <=, ...` are completely equivalent to the operators `lt, le, ...` and work in a somewhat peculiar way that first compares the length of two strings and then the lexical order. On the other hand, `ap_expr` has two sets of comparison operators: The operators `<, <=, ...` do lexical string comparison, while the operators `-lt, -le, ...` do integer comparison. For the latter, there are also aliases without the leading dashes: `lt, le, ...`

This directive specifies a general access requirement which has to be fulfilled in order to allow access. It is a very powerful directive because the requirement specification is an arbitrarily complex boolean expression containing any number of access checks.

The *expression* must match the following syntax (given as a BNF grammar notation):

```
expr ::= "true" | "false"
```
For varname any of the variables described in Environment Variables can be used. For funcname the available functions are listed in the ap_expr documentation.

The expression is parsed into an internal machine representation when the configuration is loaded, and then evaluated during request processing. In .htaccess context, the expression is both
parsed and executed each time the .htaccess file is encountered during request processing.

Example

```perl
SSLRequire (  %{SSL_CIPHER} !~ m/(EXP|NULL)/
             and %{SSL_CLIENT_S_DN_O} eq "Snake Oil, Ltd."
             and %{SSL_CLIENT_S_DN_OU} in {"Staff", "CA", "Dev"}
             and %{TIME_WDAY} -ge 1 and %{TIME_WDAY} -le 5
             and %{TIME_HOUR} -ge 8 and %{TIME_HOUR} -le 20
             or %{REMOTE_ADDR} =~ m/^192\.[76]\.[162]\.[0-9]+$/
```

The PeerExtList(object-ID) function expects to find zero or more instances of the X.509 certificate extension identified by the given object ID (OID) in the client certificate. The expression evaluates to true if the left-hand side string matches exactly against the value of an extension identified with this OID. (If multiple extensions with the same OID are present, at least one extension must match).

Example

```perl
SSLRequire "foobar" in PeerExtList("1.2.3.4.5.6")
```

Notes on the PeerExtList function

- The object ID can be specified either as a descriptive name recognized by the SSL library, such as "nsComment", or as a numeric OID, such as "1.2.3.4.5.6".

- Expressions with types known to the SSL library are rendered to a string before comparison. For an extension with a type not recognized by the SSL library, mod_ssl will parse the value if it is one of the primitive ASN.1 types UTF8String, IA5String, VisibleString, or BMPString. For an extension of one of these types, the string value will be converted to UTF-8 if necessary, then compared against
the left-hand-side expression.

See also

- Environment Variables in Apache HTTP Server, for additional examples.
- Require expr
- Generic expression syntax in Apache HTTP Server
Description: Deny access when SSL is not used for the HTTP request
Syntax: SSLRequireSSL
Context: directory, .htaccess
Override: AuthConfig
Status: Extension
Module: mod_ssl

This directive forbids access unless HTTP over SSL (i.e. HTTPS) is enabled for the current connection. This is very handy inside the SSL-enabled virtual host or directories for defending against configuration errors that expose stuff that should be protected. When this directive is present all requests are denied which are not using SSL.

Example
SSLRequireSSL
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Type of the global/inter-process SSL Session Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td>SSLSessionCache  <em>type</em></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td>SSLSessionCache  <em>none</em></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ssl</td>
</tr>
</tbody>
</table>

This configures the storage type of the global/inter-process SSL Session Cache. This cache is an optional facility which speeds up parallel request processing. For requests to the same server process (via HTTP keep-alive), OpenSSL already caches the SSL session information locally. But because modern clients request inlined images and other data via parallel requests (usually up to four parallel requests are common) those requests are served by *different* pre-forked server processes. Here an inter-process cache helps to avoid unnecessary session handshakes.

The following five storage *types* are currently supported:

- **none**
  This disables the global/inter-process Session Cache. This will incur a noticeable speed penalty and may cause problems if using certain browsers, particularly if client certificates are enabled. This setting is not recommended.

- **nonenotnull**
  This disables any global/inter-process Session Cache. However it does force OpenSSL to send a non-null session ID to accommodate buggy clients that require one.

- **dbm:/path/to/datafile**
  This makes use of a DBM hashfile on the local disk to
synchronize the local OpenSSL memory caches of the server processes. This session cache may suffer reliability issues under high load. To use this, ensure that \texttt{mod\_socache\_dbm} is loaded.

- \texttt{shmcb:/path/to/datafile[(size)]}
  This makes use of a high-performance cyclic buffer (approx. size bytes in size) inside a shared memory segment in RAM (established via /path/to/datafile) to synchronize the local OpenSSL memory caches of the server processes. This is the recommended session cache. To use this, ensure that \texttt{mod\_socache\_shmcb} is loaded.

- \texttt{dc:UNIX:/path/to/socket}
  This makes use of the \texttt{distcache} distributed session caching libraries. The argument should specify the location of the server or proxy to be used using the distcache address syntax; for example, UNIX:/path/to/socket specifies a UNIX domain socket (typically a local \texttt{dc\_client} proxy); IP:server.example.com:9001 specifies an IP address. To use this, ensure that \texttt{mod\_socache\_dc} is loaded.

### Examples

```
SSLSessionCache "dbm:/usr/local/apache/logs/ssl_gcache_data"
SSLSessionCache "shmcb:/usr/local/apache/logs/ssl_gcache_data(512000)"
```

The ssl-cache mutex is used to serialize access to the session cache to prevent corruption. This mutex can be configured using the \texttt{Mutex} directive.
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Number of seconds before an SSL session expires in the Session Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td><code>SSLSessionCacheTimeout seconds</code></td>
</tr>
<tr>
<td><strong>Default:</strong></td>
<td><code>SSLSessionCacheTimeout 300</code></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config, virtual host</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td>mod_ssl</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Applies also to RFC 5077 TLS session resumption in Apache 2.4.10 and later</td>
</tr>
</tbody>
</table>

This directive sets the timeout in seconds for the information stored in the global/inter-process SSL Session Cache, the OpenSSL internal memory cache and for sessions resumed by TLS session resumption (RFC 5077). It can be set as low as 15 for testing, but should be set to higher values like 300 in real life.

**Example**

```
SSLSessionCacheTimeout 600
```
**Description:** Persistent encryption/decryption key for TLS session tickets

**Syntax:** `SSLSessionTicketKeyFile file-path`

**Context:** server config, virtual host

**Status:** Extension

**Module:** `mod_ssl`

**Compatibility:** Available in httpd 2.4.0 and later, if using OpenSSL 0.9.8h or later

Optionally configures a secret key for encrypting and decrypting TLS session tickets, as defined in RFC 5077. Primarily suitable for clustered environments where TLS sessions information should be shared between multiple nodes. For single-instance httpd setups, it is recommended to *not* configure a ticket key file, but to rely on (random) keys generated by `mod_ssl` at startup, instead.

The ticket key file must contain 48 bytes of random data, preferrably created from a high-entropy source. On a Unix-based system, a ticket key file can be created as follows:

```
dd if=/dev/random of=/path/to/file.tkey bs=1 count=48
```

Ticket keys should be rotated (replaced) on a frequent basis, as this is the only way to invalidate an existing session ticket - OpenSSL currently doesn't allow to specify a limit for ticket lifetimes. A new ticket key only gets used after restarting the web server. All existing session tickets become invalid after a restart.

The ticket key file contains sensitive keying material and should be protected with file permissions similar to those used for `SSLCertificateKeyFile`. 
**SSLSessionTickets Directive**

**Description:** Enable or disable use of TLS session tickets

**Syntax:** `SSLSessionTickets on|off`

**Default:** `SSLSessionTickets on`

**Context:** server config, virtual host

**Status:** Extension

**Module:** `mod_ssl`

**Compatibility:** Available in httpd 2.4.11 and later, if using OpenSSL 0.9.8f or later.

This directive allows to enable or disable the use of TLS session tickets (RFC 5077).

TLS session tickets are enabled by default. Using them without restarting the web server with an appropriate frequency (e.g. daily) compromises perfect forward secrecy.
**Description:** SRP unknown user seed

**Syntax:** `SSLSRPUnknownUserSeed secret-string`

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available in httpd 2.4.4 and later, if using OpenSSL 1.0.1 or later

This directive sets the seed used to fake SRP user parameters for unknown users, to avoid leaking whether a given user exists. Specify a secret string. If this directive is not used, then Apache will return the UNKNOWN_PSK_IDENTITY alert to clients who specify an unknown username.

**Example**

`SSLSRPUnknownUserSeed "secret"`
**Description:** Path to SRP verifier file

**Syntax:**

```
SSLSRPVerifierFile file-path
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available in httpd 2.4.4 and later, if using OpenSSL 1.0.1 or later

This directive enables TLS-SRP and sets the path to the OpenSSL SRP (Secure Remote Password) verifier file containing TLS-SRP usernames, verifiers, salts, and group parameters.

**Example**

```
SSLSRPVerifierFile "/path/to/file.srpv"
```

The verifier file can be created with the `openssl` command line utility:

**Creating the SRP verifier file**

```
openssl srp -srpvfile passwd.srpv -userinfo "some info" -add username
```

The value given with the optional `-userinfo` parameter is available in the SSL_SR_P_USERINFO request environment variable.
**Description:** Configures the OCSP stapling cache

**Syntax:**
SSLStaplingCache *type*

**Context:** server config

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available if using OpenSSL 0.9.8h or later

Configures the cache used to store OCSP responses which get included in the TLS handshake if [SSLUseStapling](#) is enabled. Configuration of a cache is mandatory for OCSP stapling. With the exception of none and nonenotnull, the same storage types are supported as with [SSLSessionCache](#).
**Description:** Number of seconds before expiring invalid responses in the OCSP stapling cache

**Syntax:** `SSLStaplingErrorCacheTimeout seconds`

**Default:** `SSLStaplingErrorCacheTimeout 600`

**Context:** server config, virtual host

**Status:** Extension

**Module:** `mod_ssl`

**Compatibility:** Available if using OpenSSL 0.9.8h or later

Sets the timeout in seconds before *invalid* responses in the OCSP stapling cache (configured through `SSLStaplingCache`) will expire. To set the cache timeout for valid responses, see `SSLStaplingStandardCacheTimeout`.
**Description:** Synthesize "tryLater" responses for failed OCSP stapling queries

**Syntax:** SSLStaplingFakeTryLater on|off

**Default:** SSLStaplingFakeTryLater on

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available if using OpenSSL 0.9.8h or later

When enabled and a query to an OCSP responder for stapling purposes fails, mod_ssl will synthesize a "tryLater" response for the client. Only effective if [SSLStaplingReturnResponderErrors](#) is also enabled.
**Description:** Override the OCSP responder URI specified in the certificate's AIA extension

**Syntax:** SSLStaplingForceURL uri

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available if using OpenSSL 0.9.8h or later

This directive overrides the URI of an OCSP responder as obtained from the authorityInfoAccess (AIA) extension of the certificate. One potential use is when a proxy is used for retrieving OCSP queries.
SSLStaplingResponderTimeout

**Description:**
Timeout for OCSP stapling queries

**Syntax:**
SSLStaplingResponderTimeout *seconds*

**Default:**
SSLStaplingResponderTimeout 10

**Context:**
server config, virtual host

**Status:**
Extension

**Module:**
mod_ssl

**Compatibility:**
Available if using OpenSSL 0.9.8h or later

This option sets the timeout for queries to OCSP responders when SSLUseStapling is enabled and mod_ssl is querying a responder for OCSP stapling purposes.
**Description:** Maximum allowable age for OCSP stapling responses

**Syntax:** SSLStaplingResponseMaxAge seconds

**Default:** SSLStaplingResponseMaxAge -1

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available if using OpenSSL 0.9.8h or later

This option sets the maximum allowable age ("freshness") when considering OCSP responses for stapling purposes, i.e. when **SSLUseStapling** is turned on. The default value (-1) does not enforce a maximum age, which means that OCSP responses are considered valid as long as their nextUpdate field is in the future.
**Description:** Maximum allowable time skew for OCSP stapling response validation

**Syntax:**

SSLStaplingResponseTimeSkew seconds

**Default:**

SSLStaplingResponseTimeSkew 300

**Context:**

server config, virtual host

**Status:**

Extension

**Module:**

mod_ssl

**Compatibility:**

Available if using OpenSSL 0.9.8h or later

This option sets the maximum allowable time skew when mod_ssl checks the thisUpdate and nextUpdate fields of OCSP responses which get included in the TLS handshake (OCSP stapling). Only applicable if SSLUseStapling is turned on.
**SSLStaplingReturnResponderErrors Directive**

**Description:** Pass stapling related OCSP errors on to client

**Syntax:** SSLStaplingReturnResponderErrors on|off

**Default:** SSLStaplingReturnResponderErrors on

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available if using OpenSSL 0.9.8h or later

When enabled, mod_ssl will pass responses from unsuccessful stapling related OCSP queries (such as responses with an overall status other than "successful", responses with a certificate status other than "good", expired responses etc.) on to the client. If set to off, only responses indicating a certificate status of "good" will be included in the TLS handshake.
**Description:** Number of seconds before expiring responses in the OCSP stapling cache

**Syntax:**

```
SSLStaplingStandardCacheTimeout seconds
```

**Default:**

```
SSLStaplingStandardCacheTimeout 3600
```

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available if using OpenSSL 0.9.8h or later

Sets the timeout in seconds before responses in the OCSP stapling cache (configured through [SSLStaplingCache](https://httpd.apache.org/docs/current/mod/mod_ssl.html#sslstaplingcache)) will expire. This directive applies to *valid* responses, while [SSLStaplingErrorCacheTimeout](https://httpd.apache.org/docs/current/mod/mod_ssl.html#sslstappingerrorcachetimeout) is used for controlling the timeout for invalid/unavailable responses.
SSLStrictSNIVHostCheck

**Description:** Whether to allow non-SNI clients to access a name-based virtual host.

**Syntax:** SSLStrictSNIVHostCheck on|off

**Default:** SSLStrictSNIVHostCheck off

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available in Apache 2.2.12 and later

This directive sets whether a non-SNI client is allowed to access a name-based virtual host. If set to on in the default name-based virtual host, clients that are SNI unaware will not be allowed to access *any* virtual host, belonging to this particular IP / port combination. If set to on in any other virtual host, SNI unaware clients are not allowed to access this particular virtual host.

This option is only available if httpd was compiled against an SNI capable version of OpenSSL.

**Example**

SSLStrictSNIVHostCheck on
**Description:** Variable name to determine user name

**Syntax:** SSLUserName varname

**Context:** server config, directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_ssl

This directive sets the "user" field in the Apache request object. This is used by lower modules to identify the user with a character string. In particular, this may cause the environment variable REMOTE_USER to be set. The varname can be any of the SSL environment variables.

Note that this directive has no effect if the FakeBasicAuth option is used (see **SSLOptions**).

**Example**

SSLUserName SSL_CLIENT_S_DN_CN
**Description:** Enable stapling of OCSP responses in the TLS handshake

**Syntax:** SSLUseStapling on|off

**Default:** SSLUseStapling off

**Context:** server config, virtual host

**Status:** Extension

**Module:** mod_ssl

**Compatibility:** Available if using OpenSSL 0.9.8h or later

This option enables OCSP stapling, as defined by the "Certificate Status Request" TLS extension specified in RFC 6066. If enabled (and requested by the client), mod_ssl will include an OCSP response for its own certificate in the TLS handshake. Configuring an [SSLStaplingCache](#) is a prerequisite for enabling OCSP stapling.

OCSP stapling relieves the client of querying the OCSP responder on its own, but it should be noted that with the RFC 6066 specification, the server's CertificateStatus reply may only include an OCSP response for a single cert. For server certificates with intermediate CA certificates in their chain (the typical case nowadays), stapling in its current implementation therefore only partially achieves the stated goal of "saving roundtrips and resources" - see also [RFC 6961](https://tools.ietf.org/html/rfc6961) (TLS Multiple Certificate Status Extension).

When OCSP stapling is enabled, the ssl-stapling mutex is used to control access to the OCSP stapling cache in order to prevent corruption, and the sss-stapling-refresh mutex is used to control refreshes of OCSP responses. These mutexes can be configured using the [Mutex](#) directive.
Description: Type of Client Certificate verification

Syntax: SSLVerifyClient level

Default: SSLVerifyClient none

Context: server config, virtual host, directory, .htaccess

Override: AuthConfig

Status: Extension

Module: mod_ssl

This directive sets the Certificate verification level for the Client Authentication. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the client authentication process used in the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured client verification level after the HTTP request was read but before the HTTP response is sent.

The following levels are available for level:

- **none**: no client Certificate is required at all
- **optional**: the client may present a valid Certificate
- **require**: the client has to present a valid Certificate
- **optional_no_ca**: the client may present a valid Certificate but it need not to be (successfully) verifiable. This option cannot be relied upon for client authentication.

**Example**

SSLVerifyClient require
**Description:** Maximum depth of CA Certificates in Client Certificate verification

**Syntax:** SSLVerifyDepth number

**Default:** SSLVerifyDepth 1

**Context:** server config, virtual host, directory, .htaccess

**Override:** AuthConfig

**Status:** Extension

**Module:** mod_ssl

This directive sets how deeply mod_ssl should verify before deciding that the clients don't have a valid certificate. Notice that this directive can be used both in per-server and per-directory context. In per-server context it applies to the client authentication process used in the standard SSL handshake when a connection is established. In per-directory context it forces a SSL renegotiation with the reconfigured client verification depth after the HTTP request was read but before the HTTP response is sent.

The depth actually is the maximum number of intermediate certificate issuers, i.e. the number of CA certificates which are max allowed to be followed while verifying the client certificate. A depth of 0 means that self-signed client certificates are accepted only, the default depth of 1 means the client certificate can be self-signed or has to be signed by a CA which is directly known to the server (i.e. the CA's certificate is under SSLCACertificatePath), etc.

**Example**

SSLVerifyDepth 10
Apache mod_status

Status

- Base
- status_module
- mod_status.c

1 1
Apache CPU (*)
(*)

"(*)" ExtendedStatus On
<Location /server-status>
SetHandler server-status

Order Deny,Allow
Deny from all
Allow from .example.com
</Location>

http://your.server.name
status?refresh=N
**Apache Module mod_substitute**

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Perform search and replace operations on response bodies</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>substitute_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_substitute.c</td>
</tr>
<tr>
<td><strong>Compatibility:</strong></td>
<td>Available in Apache HTTP Server 2.2.7 and later</td>
</tr>
</tbody>
</table>

**Summary**

`mod_substitute` provides a mechanism to perform both regular expression and fixed string substitutions on response bodies.
**Description:** Pattern to filter the response content

**Syntax:** Substitute

\`s/pattern/substitution/[infq]\`

**Context:** directory, .htaccess

**Override:** FileInfo

**Status:** Extension

**Module:** mod_substitute

The **Substitute** directive specifies a search and replace pattern to apply to the response body.

The meaning of the pattern can be modified by using any combination of these flags:

**i**

Perform a case-insensitive match.

**n**

By default the pattern is treated as a regular expression. Using the n flag forces the pattern to be treated as a fixed string.

**f**

The f flag causes mod_substitute to flatten the result of a substitution allowing for later substitutions to take place on the boundary of this one. This is the default.

**q**

The q flag causes mod_substitute to not flatten the buckets after each substitution. This can result in much faster response and a decrease in memory utilization, but should only be used if there is no possibility that the result of one substitution will ever match a pattern or regex of a subsequent one.
If either the pattern or the substitution contain a slash character then an alternative delimiter should be used:

```
Example of using an alternate delimiter

<Location "/">
    AddOutputFilterByType SUBSTITUTE text/html
    Substitute "s<br */?>>|br />|i"
</Location>
```

Backreferences can be used in the comparison and in the substitution, when regular expressions are used, as illustrated in the following example:

```
Example of using backreferences and captures

<Location "/">
    AddOutputFilterByType SUBSTITUTE text/html
    # "foo=k,bar=k" -> "foo/bar=k"
    Substitute "s|foo=(\w+),bar=\1|foo/bar=$1"
</Location>
```

A common use scenario for mod_substitute is the situation in which a front-end server proxies requests to a back-end server which returns HTML with hard-coded embedded URLs that refer to the back-end server. These URLs don't work for the end-user, since the back-end server is unreachable.

In this case, mod_substitute can be used to rewrite those URLs into something that will work from the front end:
ProxyPass "/blog/" "http://internal.blog.example.com"
ProxyPassReverse "/blog/" "http://internal.blog.example.com/"
Substitute "s|http://internal.blog.example.com/|http://www.example.com/blog/|i"

**ProxyPassReverse** modifies any Location (redirect) headers that are sent by the back-end server, and, in this example, **Substitute** takes care of the rest of the problem by fixing up the HTML response as well.
**Description:** Change the merge order of inherited patterns

**Syntax:** SubstituteInheritBefore on|off

**Default:** SubstituteInheritBefore off

**Context:** directory, .htaccess

**Override:** FileInfo

**Status:** Extension

**Module:** mod_substitute

**Compatibility:** Available in httpd 2.4.17 and later

Whether to apply the inherited Substitute patterns first (on), or after the ones of the current context (off).

SubstituteInheritBefore is itself inherited, hence contexts that inherit it (those that don’t specify their own SubstituteInheritBefore value) will apply the closest defined merge order.
| **Description:** | Set the maximum line size |
| **Syntax:** | SubstituteMaxLineLength bytes(b|B|k|K|m|M|g|G) |
| **Default:** | SubstituteMaxLineLength 1m |
| **Context:** | directory, .htaccess |
| **Override:** | FileInfo |
| **Status:** | Extension |
| **Module:** | mod_substitute |
| **Compatibility:** | Available in httpd 2.4.11 and later |

The maximum line size handled by [mod substitute](https://httpd.apache.org/docs/current/mod/mod_substitute.html) is limited to restrict memory use. The limit can be configured using `SubstituteMaxLineLength`. The value can be given as the number of bytes and can be suffixed with a single letter `b`, `B`, `k`, `K`, `m`, `M`, `g`, `G` to provide the size in bytes, kilobytes, megabytes or gigabytes respectively.

**Example**

```xml
<Location />
  AddOutputFilterByType SUBSTITUTE text/html
  SubstituteMaxLineLength 10m
  Substitute "s/foo/bar/ni"
</Location>
```
Apache mod_suexec

| suexec | CGI |

**Bugfix checklist**

- [httpd changelog](httpd changelog)
- [Known issues](Known issues)
- [Report a bug](Report a bug)
SuexecUserGroup CGI

1.3 VirtualHosts User Group

SuexecUserGroup nobody nogroup
Apache mod_unique_id

: Extension
: unique_id_module
: mod_unique_id.c
Apache Unix

(NTP)

- NTP
- IP

pid (ID) 32

httpd

Unix (UTC 1970 1 1 ) 16

(ip_addr, pid, time_stamp, counter) httpd 65536

httpd (÷ 10) modulo 65536 0

httpd

rand() seed seed

? 500

1.5%

UTC

NTP UTC

UNIQUE_ID 112 (32 IP 32 pid, 32 16

[A-Za-z0-9@-] MIME base64 19

base64 [A-Za-z0-9+/] + / URL
Copyright 2017 The Apache Software Foundation.

Apache License, Version 2.0
Apache Module mod_unixd

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Basic (required) security for Unix-family platforms.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>unixd_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_unixd.c</td>
</tr>
</tbody>
</table>

**Bugfix checklist**

- [httpd changelog](#)
- [Known issues](#)
- [Report a bug](#)

**See also**

- [suEXEC support](#)
**Description:** Directory for apache to run chroot(8) after startup.

**Syntax:** ChrootDir /path/to/directory

**Default:** none

**Context:** server config

**Status:** Base

**Module:** mod_unixd

**Compatibility:** Available in Apache 2.2.10 and later

This directive tells the server to *chroot*(8) to the specified directory after startup, but before accepting requests over the 'net.

Note that running the server under chroot is not simple, and requires additional setup, particularly if you are running scripts such as CGI or PHP. Please make sure you are properly familiar with the operation of chroot before attempting to use this feature.
**Description:** Group under which the server will answer requests

**Syntax:**

Group *unix-group*

**Default:**

Group # -1

**Context:**

server config

**Status:**

Base

**Module:**

mod_unixd

The **Group** directive sets the group under which the server will answer requests. In order to use this directive, the server must be run initially as root. If you start the server as a non-root user, it will fail to change to the specified group, and will instead continue to run as the group of the original user. *Unix-group* is one of:

**A group name**

Refers to the given group by name.

**# followed by a group number.**

Refers to a group by its number.

**Example**

Group *www-group*

It is recommended that you set up a new group specifically for running the server. Some admins use user nobody, but this is not always possible or desirable.

**Security**

Don't set **Group** (or **User**) to root unless you know exactly what you are doing, and what the dangers are.

**See also**
- VHostGroup
- SuexecUserGroup
**Description:** Enable or disable the suEXEC feature

**Syntax:** Suexec On|Off

**Default:** On if suexec binary exists with proper owner and mode, Off otherwise

**Context:** server config

**Status:** Base

**Module:** mod_unixd

When On, startup will fail if the suexec binary doesn't exist or has an invalid owner or file mode.

When Off, suEXEC will be disabled even if the suexec binary exists and has a valid owner and file mode.
The `User` directive sets the user ID as which the server will answer requests. In order to use this directive, the server must be run initially as root. If you start the server as a non-root user, it will fail to change to the lesser privileged user, and will instead continue to run as that original user. If you do start the server as root, then it is normal for the parent process to remain running as root. `Unix-userid` is one of:

**A username**
- Refers to the given user by name.

**# followed by a user number.**
- Refers to a user by its number.

The user should have no privileges that result in it being able to access files that are not intended to be visible to the outside world, and similarly, the user should not be able to execute code that is not meant for HTTP requests. It is recommended that you set up a new user and group specifically for running the server. Some admins use user nobody, but this is not always desirable, since the nobody user can have other uses on the system.

**Security**
Don't set User (or Group) to root unless you know exactly what you are doing, and what the dangers are.
See also

- VHostUser
- SuexecUserGroup
Apache mod_userdir

http://example.com/~user/

Bugfix checklist

- httpd changelog
- Known issues
- Report a bug

URL
- public_html
UserDir

- disabled enabled ()
- disabled
- enabled

UserDir
http://www.example.com/~bob/one/two.html

UserDir
UserDir public_html ~bob/public_html/one/two.html
UserDir /usr/web /usr/web/bob/one/two.html
UserDir /home/*/www /home/bob/www/one/two.html

UserDir
UserDir http://www.example.com/users/bob:
http://www.example.com/users
UserDir http://www.example.com/bob/usr/0:
http://www.example.com/*/usr
UserDir http://www.example.com/~bob/one:
http://www.example.com/~*/
UserDir:

UserDir disabled
UserDir enabled user1 user2 user3

UserDir:

UserDir enabled
UserDir disabled user4 user5 user6

Userdir public_html /usr/web http://www.example.com/

http://www.example.com/~bob/one/two.html
~bob/public_html/one/two.html /usr/web/bob/one/two.html
http://www.example.com/bob/one/two.html

Apache

2.1.4 UserDir UserDir public_html
# Apache Module mod_usertrack

<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Clickstream logging of user activity on a site</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Status:</strong></td>
<td>Extension</td>
</tr>
<tr>
<td><strong>Module Identifier:</strong></td>
<td>usertrack_module</td>
</tr>
<tr>
<td><strong>Source File:</strong></td>
<td>mod_usertrack.c</td>
</tr>
</tbody>
</table>

## Summary

Provides tracking of a user through your website via browser cookies.
mod_usertrack sets a cookie which can be logged via mod_log_config configurable logging formats:

LogFormat "{%Apache}n %r %t" usertrack
CustomLog logs/clickstream.log usertrack
**Description:** The domain to which the tracking cookie applies

**Syntax:**

```
CookieDomain domain
```

**Context:**
server config, virtual host, directory, .htaccess

**Override:** FileInfo

**Status:** Extension

**Module:** mod_usertrack

This directive controls the setting of the domain to which the tracking cookie applies. If not present, no domain is included in the cookie header field.

The domain string **must** begin with a dot, and **must** include at least one embedded dot. That is, .example.com is legal, but www.example.com and .com are not.

Most browsers in use today will not allow cookies to be set for a two-part top level domain, such as .co.uk, although such a domain ostensibly fulfills the requirements above. These domains are equivalent to top level domains such as .com, and allowing such cookies may be a security risk. Thus, if you are under a two-part top level domain, you should still use your actual domain, as you would with any other top level domain (for example .example.co.uk).

```
CookieDomain .example.com
```
**Description:** Expiry time for the tracking cookie

**Syntax:**

```
CookieExpires expiry-period
```

**Context:**

server config, virtual host, directory, .htaccess

**Override:** FileInfo

**Status:** Extension

**Module:** mod_usertrack

When used, this directive sets an expiry time on the cookie generated by the usertrack module. The *expiry-period* can be given either as a number of seconds, or in the format such as "2 weeks 3 days 7 hours". Valid denominations are: years, months, weeks, days, hours, minutes and seconds. If the expiry time is in any format other than one number indicating the number of seconds, it must be enclosed by double quotes.

If this directive is not used, cookies last only for the current browser session.

```
CookieExpires "3 weeks"
```
**Description:** Name of the tracking cookie

**Syntax:** CookieName *token*

**Default:** CookieName Apache

**Context:** server config, virtual host, directory, .htaccess

**Override:** FileInfo

**Status:** Extension

**Module:** mod_usertrack

This directive allows you to change the name of the cookie this module uses for its tracking purposes. By default the cookie is named "Apache".

You must specify a valid cookie name; results are unpredictable if you use a name containing unusual characters. Valid characters include A-Z, a-z, 0-9, ",", and ".".

CookieName clicktrack
**Description:** Format of the cookie header field

**Syntax:**

- Netscape
- Cookie
- Cookie2
- RFC2109
- RFC2965

**Default:** CookieStyle Netscape

**Context:** server config, virtual host, directory, .htaccess

**Override:** FileInfo

**Status:** Extension

**Module:** mod_usertrack

This directive controls the format of the cookie header field. The three formats allowed are:

- **Netscape**, which is the original but now deprecated syntax. This is the default, and the syntax Apache has historically used.
- **Cookie** or RFC2109, which is the syntax that superseded the Netscape syntax.
- **Cookie2** or RFC2965, which is the most current cookie syntax.

Not all clients can understand all of these formats, but you should use the newest one that is generally acceptable to your users' browsers. At the time of writing, most browsers support all three of these formats, with Cookie2 being the preferred format.

**CookieStyle** Cookie2
CookieTracking Description:

Enables tracking cookie

Syntax:

CookieTracking on|off

Default:

CookieTracking off

Context:

server config, virtual host, directory, .htaccess

Override:

FileInfo

Status:

Extension

Module:

mod_usertrack

When mod_usertrack is loaded, and CookieTracking on is set, Apache will send a user-tracking cookie for all new requests. This directive can be used to turn this behavior on or off on a per-server or per-directory basis. By default, enabling mod_usertrack will not activate cookies.

CookieTracking on
Apache mod_version

httpd

<IfVersion 2.4.2>
    # current httpd version is exactly 2.4.2
</IfVersion>

<IfVersion >= 2.5>
    # use really new features :-)
</IfVersion>
<IfVersion ![operator] version> ...
</IfVersion>

, , , .htaccess

All

Extension

mod_version

<IfVersion> httpd

major[.minor[.patch]] 2.1.0 2.2

patch 0

<table>
<thead>
<tr>
<th>operator</th>
<th>httpd</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>httpd</td>
</tr>
<tr>
<td>&gt;</td>
<td>httpd</td>
</tr>
<tr>
<td>&gt;=</td>
<td>httpd</td>
</tr>
<tr>
<td>&lt;</td>
<td>httpd</td>
</tr>
<tr>
<td>&lt;=</td>
<td>httpd</td>
</tr>
</tbody>
</table>

<IfVersion >= 2.3>
  # this happens only in versions greater or
  # equal 2.3.0.
</IfVersion>

http :
<table>
<thead>
<tr>
<th>operator</th>
<th>= or ==</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>/regex/</td>
</tr>
<tr>
<td>~</td>
<td>version</td>
<td>regex</td>
</tr>
</tbody>
</table>

```xml
<IfVersion = /^2.4.[01234]$/>  
   # e.g. workaround for buggy versions
</IfVersion>

(  !):  

```xml
<IfVersion !~ ^2.4.[01234]$>
    # not for those versions
</IfVersion>

```xml
<IfVersion =  
```

Copyright 2017 The Apache Software Foundation.  
Apache License, Version 2.0 .
Apache Module mod_vhost_alias

**Description:** Provides for dynamically configured mass virtual hosting

**Status:** Extension

**Module Identifier:** vhost_alias_module

**Source File:** mod_vhost_alias.c

**Summary**

This module creates dynamically configured virtual hosts, by allowing the IP address and/or the Host: header of the HTTP request to be used as part of the pathname to determine what files to serve. This allows for easy use of a huge number of virtual hosts with similar configurations.

**Note**

If mod_alias or mod_userdir are used for translating URIs to filenames, they will override the directives of **mod_vhost_alias** described below. For example, the following configuration will map /cgi-bin/script.pl to /usr/local/apache2/cgi-bin/script.pl in all cases:

```
ScriptAlias "/cgi-bin/" "/usr/local/apache2/cgi-bin/
VirtualScriptAlias "/never/found/%0/cgi-bin/"
```

**Bugfix checklist**
httpd changelog
Known issues
Report a bug

See also
UseCanonicalName
Dynamically configured mass virtual hosting
All the directives in this module interpolate a string into a pathname. The interpolated string (henceforth called the "name") may be either the server name (see the `UseCanonicalName` directive for details on how this is determined) or the IP address of the virtual host on the server in dotted-quad format. The interpolation is controlled by specifiers inspired by `printf` which have a number of formats:

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>insert a %</td>
</tr>
<tr>
<td>%p</td>
<td>insert the port number of the virtual host</td>
</tr>
<tr>
<td>%N.M</td>
<td>insert (part of) the name</td>
</tr>
</tbody>
</table>

N and M are used to specify substrings of the name. N selects from the dot-separated components of the name, and M selects characters within whatever N has selected. M is optional and defaults to zero if it isn't present; the dot must be present if and only if M is present. The interpretation is as follows:

<table>
<thead>
<tr>
<th>Specifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>the whole name</td>
</tr>
<tr>
<td>1</td>
<td>the first part</td>
</tr>
<tr>
<td>2</td>
<td>the second part</td>
</tr>
<tr>
<td>-1</td>
<td>the last part</td>
</tr>
<tr>
<td>-2</td>
<td>the penultimate part</td>
</tr>
<tr>
<td>2+</td>
<td>the second and all subsequent parts</td>
</tr>
<tr>
<td>-2+</td>
<td>the penultimate and all preceding parts</td>
</tr>
<tr>
<td>1+ and -1+</td>
<td>the same as 0</td>
</tr>
</tbody>
</table>

If N or M is greater than the number of parts available a single underscore is interpolated.
For simple name-based virtual hosts you might use the following directives in your server configuration file:

```plaintext
UseCanonicalName Off
VirtualDocumentRoot "/usr/local/apache/vhosts/%0"
```

A request for
http://www.example.com/directory/file.html will be satisfied by the file
/usr/local/apache/vhosts/www.example.com/directory/file.html

For a very large number of virtual hosts it is a good idea to arrange the files to reduce the size of the vhosts directory. To do this you might use the following in your configuration file:

```plaintext
UseCanonicalName Off
VirtualDocumentRoot "/usr/local/apache/vhosts/%3+/%2.1/%2.2/%2.3/%2"
```

A request for
http://www.domain.example.com/directory/file.html will be satisfied by the file
/usr/local/apache/vhosts/example.com/d/o/m/domain/directory/file.html

A more even spread of files can be achieved by hashing from the end of the name, for example:

```plaintext
VirtualDocumentRoot "/usr/local/apache/vhosts/%3+/%2.-1/%2.-2/%2.-3/%2"
```

The example request would come from
/usr/local/apache/vhosts/example.com/n/i/a/domain.
Alternatively you might use:

```
VirtualDocumentRoot "/usr/local/apache/vhosts/%3+/..%
```

The example request would come from
/usr/local/apache/vhosts/example.com/d/o/m/ain/directory/file.html

A very common request by users is the ability to point multiple domains to multiple document roots without having to worry about the length or number of parts of the hostname being requested. If the requested hostname is sub.www.domain.example.com instead of simply www.domain.example.com, then using %3+ will result in the document root being
/usr/local/apache/vhosts/domain.example.com/... instead of the intended example.com directory. In such cases, it can be beneficial to use the combination %-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-%-
A request for http://www.domain.example.com/directory/file.html would be satisfied by the file /usr/local/apache/vhosts/10/20/30/40/docs/directory/file.html if the IP address of www.domain.example.com were 10.20.30.40. A request for http://www.domain.example.com/cgi-bin/script.pl would be satisfied by executing the program /usr/local/apache/vhosts/10/20/30/40/cgi-bin/script.pl.

If you want to include the . character in a VirtualDocumentRoot directive, but it clashes with a % directive, you can work around the problem in the following way:

A request for http://www.domain.example.com/directory/file.html will be satisfied by the file /usr/local/apache/vhosts/domain.example/directory/file.html.

The LogFormat directives %V and %A are useful in conjunction with this module.
**Description:** Dynamically configure the location of the document root for a given virtual host

**Syntax:**
VirtualDocumentRoot interpolated-directory|none

**Default:**
VirtualDocumentRoot none

**Context:**
server config, virtual host

**Status:**
Extension

**Module:**
mod_vhost_alias

The `VirtualDocumentRoot` directive allows you to determine where Apache HTTP Server will find your documents based on the value of the server name. The result of expanding `interpolated-directory` is used as the root of the document tree in a similar manner to the `DocumentRoot` directive’s argument. If `interpolated-directory` is none then `VirtualDocumentRoot` is turned off. This directive cannot be used in the same context as `VirtualDocumentRootIP`.

**Note**

`VirtualDocumentRoot` will override any `DocumentRoot` directives you may have put in the same context or child contexts. Putting a `VirtualDocumentRoot` in the global server scope will effectively override `DocumentRoot` directives in any virtual hosts defined later on, unless you set `VirtualDocumentRoot` to None in each virtual host.
**Description:** Dynamically configure the location of the document root for a given virtual host.

**Syntax:**

```
VirtualDocumentRootIP interpolated-directory|none
```

**Default:**

```
VirtualDocumentRootIP none
```

**Context:**

```
server config, virtual host
```

**Status:**

```
Extension
```

**Module:**

```
mod_vhost_alias
```

The **VirtualDocumentRootIP** directive is like the **VirtualDocumentRoot** directive, except that it uses the IP address of the server end of the connection for directory interpolation instead of the server name.
The VirtualScriptAlias directive allows you to determine where Apache httpd will find CGI scripts in a similar manner to VirtualDocumentRoot does for other documents. It matches requests for URIs starting /cgi-bin/, much like ScriptAlias /cgi-bin/ would.
| **Description:** | Dynamically configure the location of the CGI directory for a given virtual host |
| **Syntax:** | `VirtualScriptAliasIP interpolated-directory|none` |
| **Default:** | `VirtualScriptAliasIP none` |
| **Context:** | server config, virtual host |
| **Status:** | Extension |
| **Module:** | `mod_vhost_alias` |

The `VirtualScriptAliasIP` directive is like the `VirtualScriptAlias` directive, except that it uses the IP address of the server end of the connection for directory interpolation instead of the server name.
Apache Module mod_watchdog

| Description: | provides infrastructure for other modules to periodically run tasks |
| Status:      | Base |
| Module Identifier: | watchdog_module |
| Source File: | mod_watchdog.c |
| Compatibility: | Available in Apache 2.3 and later |

Summary

mod_watchdog defines programmatic hooks for other modules to periodically run tasks. These modules can register handlers for mod_watchdog hooks. Currently, the following modules in the Apache distribution use this functionality:

- mod_heartbeat
- mod_heartmonitor

To allow a module to use mod_watchdog functionality, mod_watchdog itself must be statically linked to the server core or, if a dynamic module, be loaded before the calling module.
**WatchdogInterval Directive**

**Description:** Watchdog interval in seconds

**Syntax:** WatchdogInterval *number-of-seconds*

**Default:** WatchdogInterval 1

**Context:** server config

**Status:** Base

**Module:** mod_watchdog

Sets the interval at which the watchdog_step hook runs. Default is to run every second.
Apache Module mod_xml2enc

<table>
<thead>
<tr>
<th>Description:</th>
<th>Enhanced charset/internationalisation support for libxml2-based filter modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status:</td>
<td>Base</td>
</tr>
<tr>
<td>Module Identifier:</td>
<td>xml2enc_module</td>
</tr>
<tr>
<td>Source File:</td>
<td>mod_xml2enc.c</td>
</tr>
<tr>
<td>Compatibility:</td>
<td>Version 2.4 and later. Available as a third-party module for 2.2.x versions</td>
</tr>
</tbody>
</table>

Summary

This module provides enhanced internationalisation support for markup-aware filter modules such as mod_proxy_html. It can automatically detect the encoding of input data and ensure they are correctly processed by the libxml2 parser, including converting to Unicode (UTF-8) where necessary. It can also convert data to an encoding of choice after markup processing, and will ensure the correct charset value is set in the HTTP Content-Type header.
There are two usage scenarios: with modules programmed to work with mod_xml2enc, and with those that are not aware of it:

**Filter modules enabled for mod_xml2enc**

Modules such as `mod_proxy_html` version 3.1 and up use the `xml2enc_charset` optional function to retrieve the charset argument to pass to the libxml2 parser, and may use the `xml2enc_filter` optional function to postprocess to another encoding. Using mod_xml2enc with an enabled module, no configuration is necessary: the other module will configure mod_xml2enc for you (though you may still want to customise it using the configuration directives below).

**Non-enabled modules**

To use it with a libxml2-based module that isn't explicitly enabled for mod_xml2enc, you will have to configure the filter chain yourself. So to use it with a filter `foo` provided by a module `mod_foo` to improve the latter's i18n support with HTML and XML, you could use

```
FilterProvider iconv xml2enc Content-Type $ 
FilterProvider iconv xml2enc Content-Type $ 
FilterProvider markup foo Content-Type $ 
FilterProvider markup foo Content-Type $ 
FilterChain iconv markup

mod_foo will now support any character set supported by either (or both) of libxml2 or apr_xlate/iconv.
Programmers writing libxml2-based filter modules are encouraged to enable them for mod_xml2enc, to provide strong i18n support for your users without reinventing the wheel. The programming API is exposed in *mod_xml2enc.h*, and a usage example is `mod_proxy_html`. 
Unlike `mod_charset_lite`, `mod_xml2enc` is designed to work with data whose encoding cannot be known in advance and thus configured. It therefore uses 'sniffing' techniques to detect the encoding of HTTP data as follows:

1. If the HTTP `Content-Type` header includes a `charset` parameter, that is used.
2. If the data start with an XML Byte Order Mark (BOM) or an XML encoding declaration, that is used.
3. If an encoding is declared in an HTML `<META>` element, that is used.
4. If none of the above match, the default value set by `xml2EncDefault` is used.

The rules are applied in order. As soon as a match is found, it is used and detection is stopped.
libxml2 always uses UTF-8 (Unicode) internally, and libxml2-based filter modules will output that by default. mod_xml2enc can change the output encoding through the API, but there is currently no way to configure that directly.

Changing the output encoding should (in theory, at least) never be necessary, and is not recommended due to the extra processing load on the server of an unnecessary conversion.
If you are working with encodings that are not supported by any of the conversion methods available on your platform, you can still alias them to a supported encoding using `xml2EncAlias`. 
<table>
<thead>
<tr>
<th><strong>Description:</strong></th>
<th>Recognise Aliases for encoding values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Syntax:</strong></td>
<td><code>xml2EncAlias charset alias [alias ...]</code></td>
</tr>
<tr>
<td><strong>Context:</strong></td>
<td>server config</td>
</tr>
<tr>
<td><strong>Status:</strong></td>
<td>Base</td>
</tr>
<tr>
<td><strong>Module:</strong></td>
<td><code>mod_xml2enc</code></td>
</tr>
</tbody>
</table>

This server-wide directive aliases one or more encoding to another encoding. This enables encodings not recognised by libxml2 to be handled internally by libxml2's encoding support using the translation table for a recognised encoding. This serves two purposes: to support character sets (or names) not recognised either by libxml2 or iconv, and to skip conversion for an encoding where it is known to be unnecessary.
**Description:** Sets a default encoding to assume when absolutely no information can be automatically detected.

**Syntax:** `xml2EncDefault name`

**Context:** server config, virtual host, directory, .htaccess

**Status:** Base

**Module:** `mod_xml2enc`

**Compatibility:** Version 2.4.0 and later; available as a third-party module for earlier versions.

If you are processing data with known encoding but no encoding information, you can set this default to help `mod_xml2enc` process the data correctly. For example, to work with the default value of Latin1 (`iso-8859-1` specified in HTTP/1.0), use

```
xml2EncDefault iso-8859-1
```
**Description:** Advise the parser to skip leading junk.

**Syntax:** `xml2StartParse element [element ...]`

**Context:** server config, virtual host, directory, .htaccess

**Status:** Base

**Module:** `mod_xml2enc`

Specify that the markup parser should start at the first instance of any of the elements specified. This can be used as a workaround where a broken backend inserts leading junk that messes up the parser (example here).

It should never be used for XML, nor well-formed HTML.
Apache 1.3 API notes

**Warning**

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

These are some notes on the Apache API and the data structures you have to deal with, *etc*. They are not yet nearly complete, but hopefully, they will help you get your bearings. Keep in mind that the API is still subject to change as we gain experience with it. (See the TODO file for what *might* be coming). However, it will be easy to adapt modules to any changes that are made. *(We have more modules to adapt than you do)*.

A few notes on general pedagogical style here. In the interest of conciseness, all structure declarations here are incomplete -- the real ones have more slots that I'm not telling you about. For the most part, these are reserved to one component of the server core or another, and should be altered by modules with caution. However, in some cases, they really are things I just haven't gotten around to yet. Welcome to the bleeding edge.

Finally, here's an outline, to give you some bare idea of what's coming up, and in what order:

- **Basic concepts.**
  - [Handlers, Modules, and Requests](#)
  - [A brief tour of a module](#)

- **How handlers work**
  - [A brief tour of the request_rec](#)
  - [Where request_rec structures come from](#)
- Handling requests, declining, and returning error codes
- Special considerations for response handlers
- Special considerations for authentication handlers
- Special considerations for logging handlers

- Resource allocation and resource pools
- Configuration, commands and the like
  - Per-directory configuration structures
  - Command handling
  - Side notes --- per-server configuration, virtual servers, etc.
We begin with an overview of the basic concepts behind the API, and how they are manifested in the code.

**Handlers, Modules, and Requests**

Apache breaks down request handling into a series of steps, more or less the same way the Netscape server API does (although this API has a few more stages than NetSite does, as hooks for stuff I thought might be useful in the future). These are:

- URI -> Filename translation
- Auth ID checking [is the user who they say they are?]
- Auth access checking [is the user authorized *here*?]
- Access checking other than auth
- Determining MIME type of the object requested
- `Fixups' -- there aren't any of these yet, but the phase is intended as a hook for possible extensions like `SetEnv', which don't really fit well elsewhere.
- Actually sending a response back to the client.
- Logging the request

These phases are handled by looking at each of a succession of *modules*, looking to see if each of them has a handler for the phase, and attempting invoking it if so. The handler can typically do one of three things:

- **Handle** the request, and indicate that it has done so by returning the magic constant OK.
- **Decline** to handle the request, by returning the magic integer constant DECLINED. In this case, the server behaves in all respects as if the handler simply hadn't been there.
- Signal an error, by returning one of the HTTP error codes. This terminates normal handling of the request, although an ErrorDocument may be invoked to try to mop up, and it will be
logged in any case.

Most phases are terminated by the first module that handles them; however, for logging, `fixups', and non-access authentication checking, all handlers always run (barring an error). Also, the response phase is unique in that modules may declare multiple handlers for it, via a dispatch table keyed on the MIME type of the requested object. Modules may declare a response-phase handler which can handle any request, by giving it the key */* (i.e., a wildcard MIME type specification). However, wildcard handlers are only invoked if the server has already tried and failed to find a more specific response handler for the MIME type of the requested object (either none existed, or they all declined).

The handlers themselves are functions of one argument (a request_rec structure. vide infra), which returns an integer, as above.

A brief tour of a module

At this point, we need to explain the structure of a module. Our candidate will be one of the messier ones, the CGI module -- this handles both CGI scripts and the ScriptAlias config file command. It's actually a great deal more complicated than most modules, but if we're going to have only one example, it might as well be the one with its fingers in every place.

Let's begin with handlers. In order to handle the CGI scripts, the module declares a response handler for them. Because of ScriptAlias, it also has handlers for the name translation phase (to recognize ScriptAlias-ed URIs), the type-checking phase (any ScriptAlias-ed request is typed as a CGI script).

The module needs to maintain some per (virtual) server information, namely, the ScriptAlias es in effect; the module
structure therefore contains pointers to a functions which builds these structures, and to another which combines two of them (in case the main server and a virtual server both have `ScriptAlias`es declared).

Finally, this module contains code to handle the `ScriptAlias` command itself. This particular module only declares one command, but there could be more, so modules have `command tables` which declare their commands, and describe where they are permitted, and how they are to be invoked.

A final note on the declared types of the arguments of some of these commands: a `pool` is a pointer to a `resource pool` structure; these are used by the server to keep track of the memory which has been allocated, files opened, `etc.`, either to service a particular request, or to handle the process of configuring itself. That way, when the request is over (or, for the configuration pool, when the server is restarting), the memory can be freed, and the files closed, `en masse`, without anyone having to write explicit code to track them all down and dispose of them. Also, a `cmd_parms` structure contains various information about the config file being read, and other status information, which is sometimes of use to the function which processes a config-file command (such as `ScriptAlias`). With no further ado, the module itself:

```c
/* Declarations of handlers. */

int translate_scriptalias (request_rec *);
int type_scriptalias (request_rec *);
int cgi_handler (request_rec *);

/* Subsidiary dispatch table for response-phase *
 * handlers, by MIME type */

handler_rec cgi_handlers[] = {
    { "application/x-httpd-cgi", cgi_handler },
    { NULL }
};
```
/* Declarations of routines to manipulate the
* module's configuration info. Note that these are
* returned, and passed in, as void *'s; the server
* core keeps track of them, but it doesn't, and can't,
* know their internal structure.
*/

void *make_cgi_server_config (pool *);
void *merge_cgi_server_config (pool *, void *, void *);

/* Declarations of routines to handle config-file commands */
extern char *script_alias(cmd_parms *, void *per_dir_config,
char *fake, char *real);

command_rec cgi_cmds[] = {
    { "ScriptAlias", script_alias, NULL, RSRC_CONF, TAKE2,
      "a fakename and a realname"},
    { NULL }
};

module cgi_module = {
    STANDARD_MODULE_STUFF,
    NULL,        /* initializer */
    NULL,        /* dir config creator */
    NULL,        /* dir merger */
    make_cgi_server_config,        /* server config */
    merge_cgi_server_config,        /* merge server config */
    cgi_cmds,        /* command table */
    cgi_handlers,    /* handlers */
    translate_scriptalias,        /* filename translation */
    NULL,        /* check_user_id */
    NULL,        /* check auth */
    NULL,        /* check access */
    type_scriptalias,        /* type_checker */
    NULL,        /* fixups */
    NULL,        /* logger */
    NULL        /* header parser */
};
The sole argument to handlers is a request_rec structure. This structure describes a particular request which has been made to the server, on behalf of a client. In most cases, each connection to the client generates only one request_rec structure.

**A brief tour of the request_rec**

The request_rec contains pointers to a resource pool which will be cleared when the server is finished handling the request; to structures containing per-server and per-connection information, and most importantly, information on the request itself.

The most important such information is a small set of character strings describing attributes of the object being requested, including its URI, filename, content-type and content-encoding (these being filled in by the translation and type-check handlers which handle the request, respectively).

Other commonly used data items are tables giving the MIME headers on the client's original request, MIME headers to be sent back with the response (which modules can add to at will), and environment variables for any subprocesses which are spawned off in the course of servicing the request. These tables are manipulated using the ap_table_get and ap_table_set routines.

Note that the Content-type header value *cannot* be set by module content-handlers using the ap_table_*() routines. Rather, it is set by pointing the content_type field in the request_rec structure to an appropriate string. *e.g.*,

```c
r->content_type = "text/html";
```
Finally, there are pointers to two data structures which, in turn, point to per-module configuration structures. Specifically, these hold pointers to the data structures which the module has built to describe the way it has been configured to operate in a given directory (via .htaccess files or `<Directory>` sections), for private data it has built in the course of servicing the request (so modules' handlers for one phase can pass `notes' to their handlers for other phases). There is another such configuration vector in the server_rec data structure pointed to by the request_rec, which contains per (virtual) server configuration data.

Here is an abridged declaration, giving the fields most commonly used:

```c
struct request_rec {
    pool *pool;
    conn_rec *connection;
    server_rec *server;

    /* What object is being requested */
    char *uri;
    char *filename;
    char *path_info;
    char *args;     /* QUERY_ARGS, if any */
    struct stat finfo;   /* Set by server core;
                           * st_mode set to zero if no such file */
    char *content_type;
    char *content_encoding;

    /* MIME header environments, in and out. Also,
     * an array containing environment variables to
     * be passed to subprocesses, so people can write
     * modules to add to that environment.
     *
     * The difference between headers_out and
     * err_headers_out is that the latter are printed
     * even on error, and persist across internal
     * redirects (so the headers printed for
     * ErrorDocument handlers will have them).
```
**Where request_rec structures come from**

Most request_rec structures are built by reading an HTTP request from a client, and filling in the fields. However, there are a few exceptions:

- If the request is to an imagemap, a type map (i.e., a *.var
file), or a CGI script which returned a local `Location:', then the resource which the user requested is going to be ultimately located by some URI other than what the client originally supplied. In this case, the server does an *internal redirect*, constructing a new request_rec for the new URI, and processing it almost exactly as if the client had requested the new URI directly.

- If some handler signaled an error, and an ErrorDocument is in scope, the same internal redirect machinery comes into play.
- Finally, a handler occasionally needs to investigate `what would happen if' some other request were run. For instance, the directory indexing module needs to know what MIME type would be assigned to a request for each directory entry, in order to figure out what icon to use.

Such handlers can construct a *sub-request*, using the functions `ap_sub_req_lookup_file`, `ap_sub_req_lookup_uri`, and `ap_sub_req_method_uri`; these construct a new request_rec structure and processes it as you would expect, up to but not including the point of actually sending a response. (These functions skip over the access checks if the sub-request is for a file in the same directory as the original request).

(Server-side includes work by building sub-requests and then actually invoking the response handler for them, via the function `ap_run_sub_req`).

**Handling requests, declining, and returning error codes**

As discussed above, each handler, when invoked to handle a
particular request_rec, has to return an int to indicate what happened. That can either be

- **OK** -- the request was handled successfully. This may or may not terminate the phase.
- **DECLINED** -- no erroneous condition exists, but the module declines to handle the phase; the server tries to find another.
- an HTTP error code, which aborts handling of the request.

Note that if the error code returned is REDIRECT, then the module should put a Location in the request's headers_out, to indicate where the client should be redirected to.

**Special considerations for response handlers**

Handlers for most phases do their work by simply setting a few fields in the request_rec structure (or, in the case of access checkers, simply by returning the correct error code). However, response handlers have to actually send a request back to the client.

They should begin by sending an HTTP response header, using the function ap_send_http_header. (You don't have to do anything special to skip sending the header for HTTP/0.9 requests; the function figures out on its own that it shouldn't do anything). If the request is marked header_only, that's all they should do; they should return after that, without attempting any further output.

Otherwise, they should produce a request body which responds to the client as appropriate. The primitives for this are ap_rputc and ap_rprintf, for internally generated output, and ap_send_fd, to copy the contents of some FILE * straight to the client.

At this point, you should more or less understand the following
piece of code, which is the handler which handles GET requests which have no more specific handler; it also shows how conditional GETs can be handled, if it's desirable to do so in a particular response handler -- `ap_set_last_modified` checks against the `If-modified-since` value supplied by the client, if any, and returns an appropriate code (which will, if nonzero, be `USE_LOCAL_COPY`). No similar considerations apply for `ap_set_content_length`, but it returns an error code for symmetry.

```c
int default_handler (request_rec *r)
{
    int errstatus;
    FILE *f;

    if (r->method_number != M_GET) return DECLINED;
    if (r->finfo.st_mode == 0) return NOT_FOUND;

    if ((errstatus = ap_set_content_length (r, r->finfo.st_size)) ||
        (errstatus = ap_set_last_modified (r, r->finfo.st_mtime)))
        return errstatus;

    f = fopen (r->filename, "r");

    if (f == NULL) {
        log_reason("file permissions deny server access", r->filename, r);
        return FORBIDDEN;
    }

    register_timeout ("send", r);
    ap_send_http_header (r);

    if (!r->header_only) send_fd (f, r);
    ap_pfclose (r->pool, f);
    return OK;
}
```

Finally, if all of this is too much of a challenge, there are a few ways out of it. First off, as shown above, a response handler which
has not yet produced any output can simply return an error code, in which case the server will automatically produce an error response. Secondly, it can punt to some other handler by invoking `ap_internal_redirect`, which is how the internal redirection machinery discussed above is invoked. A response handler which has internally redirected should always return OK.

(Invoking `ap_internal_redirect` from handlers which are not response handlers will lead to serious confusion).

**Special considerations for authentication handlers**

Stuff that should be discussed here in detail:

- Authentication-phase handlers not invoked unless auth is configured for the directory.
- Common auth configuration stored in the core per-dir configuration; it has accessors `ap_auth_type`, `ap_auth_name`, and `ap.requires`.
- Common routines, to handle the protocol end of things, at least for HTTP basic authentication (`ap_get_basic_auth_pw`, which sets the connection->user structure field automatically, and `ap_note_basic_auth_failure`, which arranges for the proper WWW-Authenticate: header to be sent back).

**Special considerations for logging handlers**

When a request has internally redirected, there is the question of what to log. Apache handles this by bundling the entire chain of redirects into a list of `request_rec` structures which are threaded through the `r->prev` and `r->next` pointers. The `request_rec` which is passed to the logging handlers in such cases is the one which was originally built for the initial request from the client; note that the `bytes_sent` field will only be correct in the last request in
the chain (the one for which a response was actually sent).
One of the problems of writing and designing a server-pool server is that of preventing leakage, that is, allocating resources (memory, open files, etc.), without subsequently releasing them. The resource pool machinery is designed to make it easy to prevent this from happening, by allowing resource to be allocated in such a way that they are *automatically* released when the server is done with them.

The way this works is as follows: the memory which is allocated, file opened, etc., to deal with a particular request are tied to a *resource pool* which is allocated for the request. The pool is a data structure which itself tracks the resources in question.

When the request has been processed, the pool is *cleared*. At that point, all the memory associated with it is released for reuse, all files associated with it are closed, and any other clean-up functions which are associated with the pool are run. When this is over, we can be confident that all the resource tied to the pool have been released, and that none of them have leaked.

Server restarts, and allocation of memory and resources for per-server configuration, are handled in a similar way. There is a *configuration pool*, which keeps track of resources which were allocated while reading the server configuration files, and handling the commands therein (for instance, the memory that was allocated for per-server module configuration, log files and other files that were opened, and so forth). When the server restarts, and has to reread the configuration files, the configuration pool is cleared, and so the memory and file descriptors which were taken up by reading them the last time are made available for reuse.

It should be noted that use of the pool machinery isn't generally obligatory, except for situations like logging handlers, where you really need to register cleanups to make sure that the log file gets
closed when the server restarts (this is most easily done by using the function `ap_pfopen`, which also arranges for the underlying file descriptor to be closed before any child processes, such as for CGI scripts, are execed), or in case you are using the timeout machinery (which isn't yet even documented here). However, there are two benefits to using it: resources allocated to a pool never leak (even if you allocate a scratch string, and just forget about it); also, for memory allocation, `ap_palloc` is generally faster than `malloc`.

We begin here by describing how memory is allocated to pools, and then discuss how other resources are tracked by the resource pool machinery.

**Allocation of memory in pools**

Memory is allocated to pools by calling the function `ap_palloc`, which takes two arguments, one being a pointer to a resource pool structure, and the other being the amount of memory to allocate (in chars). Within handlers for handling requests, the most common way of getting a resource pool structure is by looking at the pool slot of the relevant `request_rec`; hence the repeated appearance of the following idiom in module code:

```c
int my_handler(request_rec *r) {
    struct my_structure *foo;
    ...

    foo = (foo *)ap_palloc (r->pool, sizeof(my_structure));
}
```

Note that *there is no ap_pfree* -- `ap_palloc`ed memory is freed only when the associated resource pool is cleared. This means that `ap_palloc` does not have to do as much accounting as `malloc()`; all it does in the typical case is to round up the size,
bump a pointer, and do a range check.

(It also raises the possibility that heavy use of ap_palloc could cause a server process to grow excessively large. There are two ways to deal with this, which are dealt with below; briefly, you can use malloc, and try to be sure that all of the memory gets explicitly freed, or you can allocate a sub-pool of the main pool, allocate your memory in the sub-pool, and clear it out periodically. The latter technique is discussed in the section on sub-pools below, and is used in the directory-indexing code, in order to avoid excessive storage allocation when listing directories with thousands of files).

Allocating initialized memory

There are functions which allocate initialized memory, and are frequently useful. The function ap_pcalloc has the same interface as ap_palloc, but clears out the memory it allocates before it returns it. The function ap_pstrdup takes a resource pool and a char * as arguments, and allocates memory for a copy of the string the pointer points to, returning a pointer to the copy. Finally ap_pstrcat is a varargs-style function, which takes a pointer to a resource pool, and at least two char * arguments, the last of which must be NULL. It allocates enough memory to fit copies of each of the strings, as a unit; for instance:

```c
ap_pstrcat (r->pool, "foo", "/", "bar", NULL);
```

returns a pointer to 8 bytes worth of memory, initialized to "foo/bar".

Commonly-used pools in the Apache Web server

A pool is really defined by its lifetime more than anything else.
There are some static pools in http_main which are passed to various non-http_main functions as arguments at opportune times. Here they are:

**permanent_pool**
never passed to anything else, this is the ancestor of all pools

**pconf**
- subpool of permanent_pool
- created at the beginning of a config "cycle"; exists until the server is terminated or restarts; passed to all config-time routines, either via cmd->pool, or as the "pool *p" argument on those which don't take pools
- passed to the module init() functions

**ptemp**
- sorry I lie, this pool isn't called this currently in 1.3, I renamed it this in my pthreads development. I'm referring to the use of ptrans in the parent... contrast this with the later definition of ptrans in the child.
- subpool of permanent_pool
- created at the beginning of a config "cycle"; exists until the end of config parsing; passed to config-time routines via cmd->temp_pool. Somewhat of a "bastard child" because it isn't available everywhere. Used for temporary scratch space which may be needed by some config routines but which is deleted at the end of config.

**pchild**
- subpool of permanent_pool
- created when a child is spawned (or a thread is created); lives until that child (thread) is destroyed
- passed to the module child_init functions
- destruction happens right after the child_exit functions are called... (which may explain why I think child_exit is
redundant and unneeded)

**ptrans**
- should be a subpool of pchild, but currently is a subpool of permanent_pool, see above
- cleared by the child before going into the accept() loop to receive a connection
- used as connection->pool

**r->pool**
- for the main request this is a subpool of connection->pool; for subrequests it is a subpool of the parent request's pool.
- exists until the end of the request (*i.e.*, ap_destroy_sub_req, or in child_main after process_request has finished)
- note that r itself is allocated from r->pool; *i.e.*, r->pool is first created and then r is the first thing palloc()d from it

For almost everything folks do, r->pool is the pool to use. But you can see how other lifetimes, such as pchild, are useful to some modules... such as modules that need to open a database connection once per child, and wish to clean it up when the child dies.

You can also see how some bugs have manifested themself, such as setting connection->user to a value from r->pool -- in this case connection exists for the lifetime of ptrans, which is longer than r->pool (especially if r->pool is a subrequest!). So the correct thing to do is to allocate from connection->pool.

And there was another interesting bug in mod_include / mod_cgi. You'll see in those that they do this test to decide if they should use r->pool or r->main->pool. In this case the resource that they are registering for cleanup is a child process. If
it were registered in r->pool, then the code would \texttt{wait()} for
the child when the subrequest finishes. With \texttt{mod\_include} this
could be any old \#include, and the delay can be up to 3
seconds... and happened quite frequently. Instead the subprocess
is registered in r->main->pool which causes it to be cleaned up
when the entire request is done -- \textit{i.e.}, after the output has been
sent to the client and logging has happened.

\textbf{Tracking open files, etc.}

As indicated above, resource pools are also used to track other
sorts of resources besides memory. The most common are open
files. The routine which is typically used for this is \texttt{ap\_pfopen},
which takes a resource pool and two strings as arguments; the
strings are the same as the typical arguments to \texttt{fopen}, \textit{e.g.},

\begin{verbatim}
...  
FILE *f = ap_pfopen (r->pool, r->filename, "r");  
if (f == NULL) { ... } else { ... }
\end{verbatim}

There is also a \texttt{ap\_popenf} routine, which parallels the lower-level
open system call. Both of these routines arrange for the file to be
closed when the resource pool in question is cleared.

Unlike the case for memory, there \textit{are} functions to close files
allocated with \texttt{ap\_pfopen}, and \texttt{ap\_popenf}, namely
\texttt{ap\_pfclose} and \texttt{ap\_pclosef}. (This is because, on many
systems, the number of files which a single process can have
open is quite limited). It is important to use these functions to close
files allocated with \texttt{ap\_pfopen} and \texttt{ap\_popenf}, since to do
otherwise could cause fatal errors on systems such as Linux,
which react badly if the same FILE* is closed more than once.

(Using the close functions is not mandatory, since the file will
eventually be closed regardless, but you should consider it in cases where your module is opening, or could open, a lot of files).

**Other sorts of resources -- cleanup functions**

More text goes here. Describe the cleanup primitives in terms of which the file stuff is implemented; also, spawn_process.

Pool cleanups live until clear_pool() is called:
clear_pool(a) recursively calls destroy_pool() on all subpools of a; then calls all the cleanups for a; then releases all the memory for a. destroy_pool(a) calls clear_pool(a) and then releases the pool structure itself. *i.e.*, clear_pool(a) doesn't delete a, it just frees up all the resources and you can start using it again immediately.

**Fine control -- creating and dealing with sub-pools, with a note on sub-requests**

On rare occasions, too-free use of ap_palloc() and the associated primitives may result in undesirably profligate resource allocation. You can deal with such a case by creating a *sub-pool*, allocating within the sub-pool rather than the main pool, and clearing or destroying the sub-pool, which releases the resources which were associated with it. (This really *is* a rare situation; the only case in which it comes up in the standard module set is in case of listing directories, and then only with very large directories. Unnecessary use of the primitives discussed here can hair up your code quite a bit, with very little gain).

The primitive for creating a sub-pool is ap_make_sub_pool, which takes another pool (the parent pool) as an argument. When the main pool is cleared, the sub-pool will be destroyed. The sub-pool may also be cleared or destroyed at any time, by calling the functions ap_clear_pool and ap_destroy_pool, respectively.
(The difference is that ap_clear_pool frees resources associated with the pool, while ap_destroy_pool also deallocates the pool itself. In the former case, you can allocate new resources within the pool, and clear it again, and so forth; in the latter case, it is simply gone).

One final note -- sub-requests have their own resource pools, which are sub-pools of the resource pool for the main request. The polite way to reclaim the resources associated with a sub request which you have allocated (using the ap_sub_req... functions) is ap_destroy_sub_req, which frees the resource pool. Before calling this function, be sure to copy anything that you care about which might be allocated in the sub-request's resource pool into someplace a little less volatile (for instance, the filename in its request_rec structure).

(Again, under most circumstances, you shouldn't feel obliged to call this function; only 2K of memory or so are allocated for a typical sub request, and it will be freed anyway when the main request pool is cleared. It is only when you are allocating many, many sub-requests for a single main request that you should seriously consider the ap_destroy... functions).
One of the design goals for this server was to maintain external compatibility with the NCSA 1.3 server --- that is, to read the same configuration files, to process all the directives therein correctly, and in general to be a drop-in replacement for NCSA. On the other hand, another design goal was to move as much of the server's functionality into modules which have as little as possible to do with the monolithic server core. The only way to reconcile these goals is to move the handling of most commands from the central server into the modules.

However, just giving the modules command tables is not enough to divorce them completely from the server core. The server has to remember the commands in order to act on them later. That involves maintaining data which is private to the modules, and which can be either per-server, or per-directory. Most things are per-directory, including in particular access control and authorization information, but also information on how to determine file types from suffixes, which can be modified by AddType and ForceType directives, and so forth. In general, the governing philosophy is that anything which can be made configurable by directory should be; per-server information is generally used in the standard set of modules for information like Aliases and Redirects which come into play before the request is tied to a particular place in the underlying file system.

Another requirement for emulating the NCSA server is being able to handle the per-directory configuration files, generally called .htaccess files, though even in the NCSA server they can contain directives which have nothing at all to do with access control. Accordingly, after URI -> filename translation, but before performing any other phase, the server walks down the directory hierarchy of the underlying filesystem, following the translated pathname, to read any .htaccess files which might be present.
The information which is read in then has to be merged with the applicable information from the server's own config files (either from the `<Directory>` sections in access.conf, or from defaults in srm.conf, which actually behaves for most purposes almost exactly like `<Directory />`).

Finally, after having served a request which involved reading .htaccess files, we need to discard the storage allocated for handling them. That is solved the same way it is solved wherever else similar problems come up, by tying those structures to the per-transaction resource pool.

**Per-directory configuration structures**

Let's look out how all of this plays out in mod_mime.c, which defines the file typing handler which emulates the NCSA server's behavior of determining file types from suffixes. What we'll be looking at, here, is the code which implements the AddType and AddEncoding commands. These commands can appear in .htaccess files, so they must be handled in the module's private per-directory data, which in fact, consists of two separate tables for MIME types and encoding information, and is declared as follows:

```c
typedef struct {
    table *forced_types; /* Additional AddTyped stuff */
    table *encoding_types; /* Added with AddEncoding... */
} mime_dir_config;
```

When the server is reading a configuration file, or `<Directory>` section, which includes one of the MIME module's commands, it needs to create a mime_dir_config structure, so those commands have something to act on. It does this by invoking the function it finds in the module's `create per-dir config slot`, with two arguments: the name of the directory to which this configuration
information applies (or NULL for srm.conf), and a pointer to a resource pool in which the allocation should happen.

(If we are reading a .htaccess file, that resource pool is the per-request resource pool for the request; otherwise it is a resource pool which is used for configuration data, and cleared on restarts. Either way, it is important for the structure being created to vanish when the pool is cleared, by registering a cleanup on the pool if necessary).

For the MIME module, the per-dir config creation function just ap_pallocs the structure above, and a creates a couple of tables to fill it. That looks like this:

```c
void *create_mime_dir_config (pool *p, char *dummy)
{
    mime_dir_config *new =
        (mime_dir_config *) ap_palloc (p,
        sizeof(mime_dir_config));

    new->forced_types = ap_make_table (p, 4);
    new->encoding_types = ap_make_table (p, 4);

    return new;
}
```

Now, suppose we've just read in a .htaccess file. We already have the per-directory configuration structure for the next directory up in the hierarchy. If the .htaccess file we just read in didn't have any AddType or AddEncoding commands, its per-directory config structure for the MIME module is still valid, and we can just use it. Otherwise, we need to merge the two structures somehow.

To do that, the server invokes the module's per-directory config merge function, if one is present. That function takes three arguments: the two structures being merged, and a resource pool in which to allocate the result. For the MIME module, all that needs
to be done is overlay the tables from the new per-directory config structure with those from the parent:

```c
void *merge_mime_dir_configs (pool *p, void *parent_dirv, void *subdirv)
{
    mime_dir_config *parent_dir = (mime_dir_config *)parent_dirv;
    mime_dir_config *subdir = (mime_dir_config *)subdirv;
    mime_dir_config *new = (mime_dir_config *)ap_palloc (p, sizeof(mime_dir_config));

    new->forced_types = ap_overlay_tables (p, subdir->forced_types,
                                           parent_dir->forced_types);
    new->encoding_types = ap_overlay_tables (p, subdir->encoding_types,
                                            parent_dir->encoding_types);

    return new;
}
```

As a note -- if there is no per-directory merge function present, the server will just use the subdirectory's configuration info, and ignore the parent's. For some modules, that works just fine (e.g., for the includes module, whose per-directory configuration information consists solely of the state of the XBITHACK), and for those modules, you can just not declare one, and leave the corresponding structure slot in the module itself NULL.

**Command handling**

Now that we have these structures, we need to be able to figure out how to fill them. That involves processing the actual `AddType` and `AddEncoding` commands. To find commands, the server looks in the module's command table. That table contains information on how many arguments the commands take, and in what formats, where it is permitted, and so forth. That information is sufficient to allow the server to invoke most command-handling functions with pre-parsed arguments. Without further ado, let's
look at the \texttt{AddType} command handler, which looks like this (the \texttt{AddEncoding} command looks basically the same, and won't be shown here):

\begin{verbatim}
char *add_type(cmd_parms *cmd, mime_dir_config *m, char *ct, char *ext)
{
  if (*ext == '.') ++ext;
  ap_table_set (m->forced_types, ext, ct);
  return NULL;
}
\end{verbatim}

This command handler is unusually simple. As you can see, it takes four arguments, two of which are pre-parsed arguments, the third being the per-directory configuration structure for the module in question, and the fourth being a pointer to a cmd_parms structure. That structure contains a bunch of arguments which are frequently of use to some, but not all, commands, including a resource pool (from which memory can be allocated, and to which cleanups should be tied), and the (virtual) server being configured, from which the module's per-server configuration data can be obtained if required.

Another way in which this particular command handler is unusually simple is that there are no error conditions which it can encounter. If there were, it could return an error message instead of \texttt{NULL}; this causes an error to be printed out on the server's stderr, followed by a quick exit, if it is in the main config files; for a .htaccess file, the syntax error is logged in the server error log (along with an indication of where it came from), and the request is bounced with a server error response (HTTP error status, code 500).

The MIME module's command table has entries for these commands, which look like this:
The entries in these tables are:

- The name of the command
- The function which handles it
- A `(void *)` pointer, which is passed in the `cmd_parms` structure to the command handler --- this is useful in case many similar commands are handled by the same function.
- A bit mask indicating where the command may appear. There are mask bits corresponding to each `AllowOverride` option, and an additional mask bit, `RSRC_CONF`, indicating that the command may appear in the server's own config files, but *not* in any `.htaccess` file.
- A flag indicating how many arguments the command handler wants pre-parsed, and how they should be passed in. `TAKE2` indicates two pre-parsed arguments. Other options are `TAKE1`, which indicates one pre-parsed argument, `FLAG`, which indicates that the argument should be `On` or `Off`, and is passed in as a boolean flag, `RAW_ARGS`, which causes the server to give the command the raw, unparsed arguments (everything but the command name itself). There is also `ITERATE`, which means that the handler looks the same as `TAKE1`, but that if multiple arguments are present, it should be called multiple times, and finally `ITERATE2`, which indicates that the command handler looks like a `TAKE2`, but if more arguments are present, then it should be called multiple times, holding the first argument constant.
Finally, we have a string which describes the arguments that should be present. If the arguments in the actual config file are not as required, this string will be used to help give a more specific error message. (You can safely leave this NULL).

Finally, having set this all up, we have to use it. This is ultimately done in the module's handlers, specifically for its file-typing handler, which looks more or less like this; note that the per-directory configuration structure is extracted from the request_rec's per-directory configuration vector by using the ap_get_module_config function.

```c
int find_ct(request_rec *r)
{
    int i;
    char *fn = ap_pstrdup(r->pool, r->filename);
    mime_dir_config *conf = (mime_dir_config *)ap_get_module_config(r->per_dir_config, &mime_module);
    char *type;

    if (S_ISDIR(r->finfo.st_mode)) {
        r->content_type = DIR_MAGIC_TYPE;
        return OK;
    }

    if ((i=ap_rind(fn,'.')) < 0) return DECLINED;
    ++i;

    if ((type = ap_table_get (conf->encoding_types, &fn[i])))
    {
        r->content_encoding = type;

        /* go back to previous extension to try to use it as a type */
        fn[i-1] = '\0';
        if((i=ap_rind(fn,'.')) < 0) return OK;
        ++i;
    }

    if ((type = ap_table_get (conf->forced_types, &fn[i])))
    {
        r->content_type = type;
    }
```
Side notes -- per-server configuration, virtual servers, etc.

The basic ideas behind per-server module configuration are basically the same as those for per-directory configuration; there is a creation function and a merge function, the latter being invoked where a virtual server has partially overridden the base server configuration, and a combined structure must be computed. (As with per-directory configuration, the default if no merge function is specified, and a module is configured in some virtual server, is that the base configuration is simply ignored).

The only substantial difference is that when a command needs to configure the per-server private module data, it needs to go to the cmd_parms data to get at it. Here's an example, from the alias module, which also indicates how a syntax error can be returned (note that the per-directory configuration argument to the command handler is declared as a dummy, since the module doesn't actually have per-directory config data):

```c
char *add_redirect(cmd_parms *cmd, void *dummy, char *f, char *url)
{
    server_rec *s = cmd->server;
    alias_server_conf *conf = (alias_server_conf *)
        ap_get_module_config(s->module_config,&alias_module);
    alias_entry *new = ap_push_array (conf->redirects);

    if (!ap_is_url (url)) return "Redirect to non-URL";

    new->fake = f; new->real = url;
    return NULL;
}
```
Debugging Memory Allocation in APR

This document has been removed.
Documenting code in Apache 2.4

Apache 2.4 uses Doxygen to document the APIs and global variables in the code. This will explain the basics of how to document using Doxygen.
To start a documentation block, use /**
To end a documentation block, use */

In the middle of the block, there are multiple tags we can use:

Description of this functions purpose
@param parameter_name description
@return description
@deffunc signature of the function

The deffunc is not always necessary. DoxyGen does not have a full parser in it, so any prototype that use a macro in the return type declaration is too complex for scandoc. Those functions require a deffunc. An example (using &gt; rather than >):

/**
 * return the final element of the pathname
 * @param pathname The path to get the final element of
 * @return the final element of the path
 * @tip Examples:
 * <pre>
 *="/foo/bar/gum" &gt; "gum"
 *="/foo/bar/gum/" &gt; ""
 * "gum" &gt; "gum"
 * "wi\n32\stuff" &gt; "stuff"
 * </pre>
 * @deffunc const char * ap_filename_of_pathname(const char *pathname)
 */

At the top of the header file, always include:

/**
 * @package Name of library header
 */

Doxygen uses a new HTML file for each package. The HTML files are named {Name_of_library_header}.html, so try to be concise with your names.
For a further discussion of the possibilities please refer to the Doxygen site.
Hook Functions in the Apache HTTP Server 2.x

Warning
This document is still in development and may be partially out of date.

In general, a hook function is one that the Apache HTTP Server will call at some point during the processing of a request. Modules can provide functions that are called, and specify when they get called in comparison to other modules.
The httpd's core modules offer a predefined list of hooks used during the standard request processing phase. Creating a new hook will expose a function that implements it (see sections below) but it is essential to understand that you will not extend the httpd's core hooks. Their presence and order in the request processing is in fact a consequence of how they are called in server/request.c (check this section for an overview). The core hooks are listed in the doxygen documentation.

Reading guide for developing modules and request processing before proceeding is highly recommended.
In order to create a new hook, four things need to be done:

**Declare the hook function**

Use the AP_DECLARE_HOOK macro, which needs to be given the return type of the hook function, the name of the hook, and the arguments. For example, if the hook returns an int and takes a request_rec * and an int and is called do_something, then declare it like this:

```c
AP_DECLARE_HOOK(int, do_something, (request_rec *r, int n))
```

This should go in a header which modules will include if they want to use the hook.

**Create the hook structure**

Each source file that exports a hook has a private structure which is used to record the module functions that use the hook. This is declared as follows:

```c
APR_HOOK_STRUCT(
    APR_HOOK_LINK(do_something)
    ...
)
```

**Implement the hook caller**

The source file that exports the hook has to implement a function that will call the hook. There are currently three possible ways to do this. In all cases, the calling function is called `ap_run_hoookname()`. 
**Void hooks**

If the return value of a hook is `void`, then all the hooks are called, and the caller is implemented like this:

```c
AP_IMPLEMENT_HOOK_VOID(do_something, (request_rec *r, int n), (r, n))
```

The second and third arguments are the dummy argument declaration and the dummy arguments as they will be used when calling the hook. In other words, this macro expands to something like this:

```c
void ap_run_do_something(request_rec *r, int n) {
    ...
    do_something(r, n);
}
```

**Hooks that return a value**

If the hook returns a value, then it can either be run until the first hook that does something interesting, like so:

```c
AP_IMPLEMENT_HOOK_RUN_FIRST(int, do_something, (request_rec *r, int n), (r, n), DECLINED)
```

The first hook that does *not* return `DECLINED` stops the loop and its return value is returned from the hook caller. Note that `DECLINED` is the traditional hook return value meaning "I didn't do anything", but it can be whatever suits you.

Alternatively, all hooks can be run until an error occurs. This boils down to permitting *two* return values, one of which means "I did something, and it was OK" and the other meaning "I did nothing".
The first function that returns a value other than one of those two stops the loop, and its return is the return value. Declare these like so:

```c
AP_IMPLEMENT_HOOK_RUN_ALL(int, do_something

Again, OK and DECLINED are the traditional values. You can use what you want.

**Call the hook callers**

At appropriate moments in the code, call the hook caller, like so:

```c
int n, ret;
request_rec *r;
ret=ap_run_do_something(r, n);
```
A module that wants a hook to be called needs to do two things.

**Implement the hook function**

Include the appropriate header, and define a static function of the correct type:

```c
static int my_something_doer(request_rec *r)
{
    ...
    return OK;
}
```

**Add a hook registering function**

During initialisation, the server will call each modules hook registering function, which is included in the module structure:

```c
static void my_register_hooks()
{
    ap_hook_do_something(my_something_doer,
}

mode MODULE_VAR_EXPORT my_module =
{
    ...
    my_register_hooks /* register hook */
};
```

**Controlling hook calling order**

In the example above, we didn't use the three arguments in the hook registration function that control calling order of all the
functions registered within the hook. There are two mechanisms for doing this. The first, rather crude, method, allows us to specify roughly where the hook is run relative to other modules. The final argument control this. There are three possible values: APR_HOOK_FIRST, APR_HOOK_MIDDLE and APR_HOOK_LAST.

All modules using any particular value may be run in any order relative to each other, but, of course, all modules using APR_HOOK_FIRST will be run before APR_HOOK_MIDDLE which are before APR_HOOK_LAST. Modules that don't care when they are run should use APR_HOOK_MIDDLE. These values are spaced out, so that positions like APR_HOOK_FIRST-2 are possible to hook slightly earlier than other functions.

Note that there are two more values, APR_HOOK_REALLY_FIRST and APR_HOOK_REALLY_LAST. These should only be used by the hook exporter.

The other method allows finer control. When a module knows that it must be run before (or after) some other modules, it can specify them by name. The second (third) argument is a NULL-terminated array of strings consisting of the names of modules that must be run before (after) the current module. For example, suppose we want "mod_xyz.c" and "mod_abc.c" to run before we do, then we'd hook as follows:

```c
static void register_hooks()
{
    static const char * const aszPre[] = {
        "mod_xyz.c",
        "mod_abc.c",
        NULL
    };

    ap_hook_do_something(my_something_doer, aszPre, NULL, APR_HOOK_MIDDLE);
}
```

Note that the sort used to achieve this is stable, so ordering set by
APR_HOOK_ORDER is preserved, as far as is possible.
mod_mmap_static Apache 2.0
apr_status_t apr_status_t
ARP_SUCCESS

- apr_pool_t *p
- apr_pool_t *plog
- apr_pool_t *ptemp
- server_rec *s

**APR**

- pool becomes apr_pool_t
- table becomes apr_table_t
mod_mmap_static:

```c
static void register_hooks(void)
{
    static const char * const aszPre[]={ "http_core.c", NULL };
    ap_hook_post_config(mmap_post_config, NULL, NULL, HOOK_MIDDLE);
    ap_hook_translate_name(mmap_static_xlat, aszPre, NULL, HOOK_LAST);
}
```

post_config ( ?

```c
ap_hook_phase_name(function_name, predecessors, successors, position);
```

...

- HOOK_FIRST
- HOOK_MIDDLE
- HOOK_LAST

```
mod_mmap_static post_config
mmap_static_xlat core aszPre
```
module MODULE_VAR_EXPORT module_name_module =
{
  STANDARD_MODULE_STUFF,
  /* initializer */
  /* dir config creator */
  /* dir merger --- default is to override */
  /* server config */
  /* merge server config */
  /* command handlers */
  /* handlers */
  /* filename translation */
  /* check_user_id */
  /* check auth */
  /* check access */
  /* type_checker */
  /* fixups */
  /* logger */
  /* header parser */
  /* child_init */
  /* child_exit */
  /* post read-request */
};

...
ap_hook_post_config
  ( _init)
ap_hook_http_method
  (HTTP ()
ap_hook_open_logs
  ()
ap_hook_auth_checker
  ()
ap_hook_access_checker
  ()
ap_hook_check_user_id
  (ID )
ap_hook_default_port
  ()
ap_hook_pre_connection
  ( accept)
ap_hook_process_connection
  ()
ap_hook_child_init
   ()
ap_hook_create_request
   (??)
ap_hook_fixups
   ()
ap_hook_handler
   ()
ap_hook_header_parser
   (post_read_request )
ap_hook_insert_filter
   ()
ap_hook_log_transaction
   ()
ap_hook_optional_fn_retrieve
   ()
ap_hook_post_read_request
   ()
ap_hook_quick_handler

ap_hook_translate_name
   (URI )
ap_hook_type_checker
   ()
Several changes in 2.0 and above affect the internal request processing mechanics. Module authors need to be aware of these changes so they may take advantage of the optimizations and security enhancements.

The first major change is to the subrequest and redirect mechanisms. There were a number of different code paths in the Apache HTTP Server 1.3 to attempt to optimize subrequest or redirect behavior. As patches were introduced to 2.0, these optimizations (and the server behavior) were quickly broken due to this duplication of code. All duplicate code has been folded back into `ap_process_request_internal()` to prevent the code from falling out of sync again.

This means that much of the existing code was 'unoptimized'. It is the Apache HTTP Project's first goal to create a robust and correct implementation of the HTTP server RFC. Additional goals include security, scalability and optimization. New methods were sought to optimize the server (beyond the performance of 1.3) without introducing fragile or insecure code.
All requests pass through `ap_process_request_internal()` in `server/request.c`, including subrequests and redirects. If a module doesn't pass generated requests through this code, the author is cautioned that the module may be broken by future changes to request processing.

To streamline requests, the module author can take advantage of the hooks offered to drop out of the request cycle early, or to bypass core hooks which are irrelevant (and costly in terms of CPU.)
Unescapes the URL

The request's parsed_uri path is unescaped, once and only once, at the beginning of internal request processing.

This step is bypassed if the proxyreq flag is set, or the parsed_uri.path element is unset. The module has no further control of this one-time unescape operation, either failing to unescape or multiply unescaping the URL leads to security repercussions.

Strips Parent and This Elements from the URI

All ../ and ./ elements are removed by ap_getparents(). This helps to ensure the path is (nearly) absolute before the request processing continues.

This step cannot be bypassed.

Initial URI Location Walk

Every request is subject to an ap_location_walk() call. This ensures that <Location> sections are consistently enforced for all requests. If the request is an internal redirect or a sub-request, it may borrow some or all of the processing from the previous or parent request's ap_location_walk, so this step is generally very efficient after processing the main request.

translate_name

Modules can determine the file name, or alter the given URI in this step. For example, mod_vhost_alias will translate the URI's path into the configured virtual host, mod_alias will translate the path to an alias path, and if the request falls back on the core, the
DocumentRoot is prepended to the request resource.

If all modules DECLINE this phase, an error 500 is returned to the browser, and a "couldn't translate name" error is logged automatically.

**Hook: map_to_storage**

After the file or correct URI was determined, the appropriate per-dir configurations are merged together. For example, mod_proxy compares and merges the appropriate <Proxy> sections. If the URI is nothing more than a local (non-proxy) TRACE request, the core handles the request and returns DONE. If no module answers this hook with OK or DONE, the core will run the request filename against the <Directory> and <Files> sections. If the request 'filename' isn't an absolute, legal filename, a note is set for later termination.

**URI Location Walk**

Every request is hardened by a second ap_location_walk() call. This reassures that a translated request is still subjected to the configured <Location> sections. The request again borrows some or all of the processing from its previous location_walk above, so this step is almost always very efficient unless the translated URI mapped to a substantially different path or Virtual Host.

**Hook: header_parser**

The main request then parses the client's headers. This prepares the remaining request processing steps to better serve the client's request.
Needs Documentation. Code is:

```c
if ((access_status = ap_run_access_checker(r)) != 0)
    return decl_die(access_status, "check access");

if ((access_status = ap_run_check_user_id(r)) != 0)
    return decl_die(access_status, "check user");

if ((access_status = ap_run_auth_checker(r)) != 0)
    return decl_die(access_status, "check authorization");
```
**Hook: type_checker**

The modules have an opportunity to test the URI or filename against the target resource, and set mime information for the request. Both `mod_mime` and `mod_mime_magic` use this phase to compare the file name or contents against the administrator's configuration and set the content type, language, character set and request handler. Some modules may set up their filters or other request handling parameters at this time.

If all modules DECLINE this phase, an error 500 is returned to the browser, and a "couldn't find types" error is logged automatically.

**Hook: fixups**

Many modules are 'trounced' by some phase above. The fixups phase is used by modules to 'reassert' their ownership or force the request's fields to their appropriate values. It isn't always the cleanest mechanism, but occasionally it's the only option.
This phase is **not** part of the processing in `ap_process_request_internal()`. Many modules prepare one or more subrequests prior to creating any content at all. After the core, or a module calls `ap_process_request_internal()` it then calls `ap_invoke_handler()` to generate the request.

**Hook: insert_filter**

Modules that transform the content in some way can insert their values and override existing filters, such that if the user configured a more advanced filter out-of-order, then the module can move its order as need be. There is no result code, so actions in this hook better be trusted to always succeed.

**Hook: handler**

The module finally has a chance to serve the request in its handler hook. Note that not every prepared request is sent to the handler hook. Many modules, such as `mod_autoindex`, will create subrequests for a given URI, and then never serve the subrequest, but simply lists it for the user. Remember not to put required teardown from the hooks above into this module, but register pool cleanups against the request pool to free resources as required.
How filters work in Apache 2.0

Warning
This is a cut 'n paste job from an email (<022501c1c529$f63a9550$7f00000a@KOJ>) and only reformatted for better readability. It's not up to date but may be a good start for further research.
There are three basic filter types (each of these is actually broken down into two categories, but that comes later).

**CONNECTION**
Filters of this type are valid for the lifetime of this connection. (AP_FTYPE_CONNECTION, AP_FTYPE_NETWORK)

**PROTOCOL**
Filters of this type are valid for the lifetime of this request from the point of view of the client, this means that the request is valid from the time that the request is sent until the time that the response is received. (AP_FTYPE_PROTOCOL, AP_FTYPE_TRANSCODE)

**RESOURCE**
Filters of this type are valid for the time that this content is used to satisfy a request. For simple requests, this is identical to PROTOCOL, but internal redirects and sub-requests can change the content without ending the request. (AP_FTYPE_Resource, AP_FTYPE_CONTENT_SET)

It is important to make the distinction between a protocol and a resource filter. A resource filter is tied to a specific resource, it may also be tied to header information, but the main binding is to a resource. If you are writing a filter and you want to know if it is resource or protocol, the correct question to ask is: "Can this filter be removed if the request is redirected to a different resource?" If the answer is yes, then it is a resource filter. If it is no, then it is most likely a protocol or connection filter. I won't go into connection filters, because they seem to be well understood. With this definition, a few examples might help:

**Byterange**
We have coded it to be inserted for all requests, and it is removed if not used. Because this filter is active at the
beginning of all requests, it can not be removed if it is redirected, so this is a protocol filter.

http_header
This filter actually writes the headers to the network. This is obviously a required filter (except in the asis case which is special and will be dealt with below) and so it is a protocol filter.

Deflate
The administrator configures this filter based on which file has been requested. If we do an internal redirect from an autoindex page to an index.html page, the deflate filter may be added or removed based on config, so this is a resource filter.

The further breakdown of each category into two more filter types is strictly for ordering. We could remove it, and only allow for one filter type, but the order would tend to be wrong, and we would need to hack things to make it work. Currently, the RESOURCE filters only have one filter type, but that should change.
This is actually rather simple in theory, but the code is complex. First of all, it is important that everybody realize that there are three filter lists for each request, but they are all concatenated together:

- `r->output_filters` (corresponds to RESOURCE)
- `r->proto_output_filters` (corresponds to PROTOCOL)
- `r->connection->output_filters` (corresponds to CONNECTION)

The problem previously, was that we used a singly linked list to create the filter stack, and we started from the "correct" location. This means that if I had a RESOURCE filter on the stack, and I added a CONNECTION filter, the CONNECTION filter would be ignored. This should make sense, because we would insert the connection filter at the top of the `c->output_filters` list, but the end of `r->output_filters` pointed to the filter that used to be at the front of `c->output_filters`. This is obviously wrong.

The new insertion code uses a doubly linked list. This has the advantage that we never lose a filter that has been inserted. Unfortunately, it comes with a separate set of headaches.

The problem is that we have two different cases were we use subrequests. The first is to insert more data into a response. The second is to replace the existing response with an internal redirect. These are two different cases and need to be treated as such.

In the first case, we are creating the subrequest from within a handler or filter. This means that the next filter should be passed to `make_sub_request` function, and the last resource filter in the sub-request will point to the next filter in the main request. This makes sense, because the sub-request's data needs to flow through the same set of filters as the main request. A graphical
representation might help:

```plaintext
Default_handler --> includes_filter -->byterange --> ...
```

If the includes filter creates a sub request, then we don't want the data from that sub-request to go through the includes filter, because it might not be SSI data. So, the subrequest adds the following:

```plaintext
Default_handler --> includes_filter -->byterange --> ...
/  
Default_handler --> sub_request_core
```

What happens if the subrequest is SSI data? Well, that's easy, the includes_filter is a resource filter, so it will be added to the sub request in between the Default_handler and the sub_request_core filter.

The second case for sub-requests is when one sub-request is going to become the real request. This happens whenever a sub-request is created outside of a handler or filter, and NULL is passed as the next filter to the make_sub_request function.

In this case, the resource filters no longer make sense for the new request, because the resource has changed. So, instead of starting from scratch, we simply point the front of the resource filters for the sub-request to the front of the protocol filters for the old request. This means that we won't lose any of the protocol filters, neither will we try to send this data through a filter that shouldn't see it.

The problem is that we are using a doubly-linked list for our filter stacks now. But, you should notice that it is possible for two lists to intersect in this model. So, you do you handle the previous pointer? This is a very difficult question to answer, because there
is no "right" answer, either method is equally valid. I looked at why we use the previous pointer. The only reason for it is to allow for easier addition of new servers. With that being said, the solution I chose was to make the previous pointer always stay on the original request.

This causes some more complex logic, but it works for all cases. My concern in having it move to the sub-request, is that for the more common case (where a sub-request is used to add data to a response), the main filter chain would be wrong. That didn't seem like a good idea to me.
The final topic. :-) Mod_Asis is a bit of a hack, but the handler needs to remove all filters except for connection filters, and send the data. If you are using mod_asis, all other bets are off.
The absolutely last point is that the reason this code was so hard to get right, was because we had hacked so much to force it to work. I wrote most of the hacks originally, so I am very much to blame. However, now that the code is right, I have started to remove some hacks. Most people should have seen that the `reset_filters` and `add_required_filters` functions are gone. Those inserted protocol level filters for error conditions, in fact, both functions did the same thing, one after the other, it was really strange. Because we don't lose protocol filters for error cases any more, those hacks went away. The HTTP_HEADER, Content-length, and Byterange filters are all added in the `insert_filters` phase, because if they were added earlier, we had some interesting interactions. Now, those could all be moved to be inserted with the HTTP_IN, CORE, and CORE_IN filters. That would make the code easier to follow.
Apache
Apache
:

APache eXtension Tool (apxs)
( DSO)
:: apxs

Apache Portable Runtime (APR)
(: Apache HTTP Server) OS
Apache HTTP Server
:: Apache Portable Runtime

::

: CA
:: SSL/TLS
(: Certificate Signing Request) (CSR)
:: CA

:: SSL/TLS
(: Certification Authority) (CA)
:: SSL/TLS

Cipher
    DES, IDEA, RC4
:: SSL/TLS
<table>
<thead>
<tr>
<th>Cipher</th>
<th>SSL/TLS</th>
</tr>
</thead>
</table>

**Common Gateway Interface (CGI)**

<table>
<thead>
<tr>
<th>CGI</th>
<th>Apache httpd</th>
</tr>
</thead>
</table>

CONNECT

<table>
<thead>
<tr>
<th>HTTP HTTP SSL</th>
<th>Apache httpd (env-variable)</th>
</tr>
</thead>
</table>

Apache

Dynamic Shared Object) **(DSO)**

Apache httpd (env-variable)

Export Administration Regulations (EAR)

Export-Crippled)
SSL/TLS

Fully-Qualified Domain-Name (FQDN) IP
www.example.com

Apache CGI

Apache

.http

.httpd.conf

Apache /usr/local/apache2/conf/httpd.conf

HyperText Transfer Protocol (HTTP)
World Wide Web Apache HTTP/1.1

HTTPS
The HyperText Transfer Protocol (Secure), World Wide Web SSL HTTP
HTTP

: SSL/TLS

MIME

Multipurpose Internet Mail Extensions

text/html, image/gif, application/octet-stream

HTTP MIME

: mod_mime

Apache

Apache HTTP

: tarball Apache

Module Magic Number (MMN)

Apache

Apache

OpenSSL

SSL/TLS

http://www.openssl.org/

: SSL/TLS

: mod_proxy
Secure Sockets Layer (SSL)
Netscape Communications Corporation TCP/IP
HTTPS SSL HyperText Transfer Protocol (HTTP)

Server Side Includes (SSI)
HTML

SSLeay
Eric A. Young SSL/TLS

Cipher

Tarball
tar Apache tai

Transport Layer Security (TLS)
TCP/IP Internet Engineering Task Force (IETF)
TLS 1 SSL 3
:  SSL/TLS

**Uniform Resource Locator (URL)**
Internet
http  https  URL
http://httpd.apache.org/docs/2.4/glossary.html

**Uniform Resource Identifier (URI)**

Apache
:  Apache

X.509
SSL/TLS  International  Telecommunication Union
:  SSL/TLS
Apache

- AcceptFilter
- AcceptPathInfo
- AccessFileName
- Action
- AddAlt
- AddAltByEncoding
- AddAltByType
- AddCharset
- AddDefaultCharset
- AddDescription
- AddEncoding
- AddHandler
- AddIcon
- AddIconByEncoding
- AddIconByType
- AddInputFilter
- AddLanguage
- AddModuleInfo
- AddOutputFilter
- AddOutputFilterByType
- AddType
- Alias
- AliasMatch
- Allow
- AllowCONNECT
- AllowEncodedSlashes
- AllowMethods
- AllowOverride
- AllowOverrideList
- Anonymous
- Anonymous_LogEmail
- Anonymous_MustGiveEmail
- Anonymous_NoUserID
- Anonymous_VerifyEmail
- AsyncRequestWorkerFactor
- AuthBasicAuthoritative
- AuthBasicFake
- AuthBasicProvider
- AuthBasicUseDigestAlgorithm
- AuthDBDUserPWQuery
- AuthDBDUserRealmQuery
- AuthDBMGroupFile
- AuthDBMType
- AuthDBMUserFile
- AuthDigestAlgorithm
- AuthDigestDomain
- AuthDigestNonceLifetime
- AuthDigestProvider
- AuthDigestQop
- AuthDigestShmemSize
- AuthFormAuthoritative
- AuthFormBody
- AuthFormDisableNoStore
- AuthFormFakeBasicAuth
- AuthFormLocation
- AuthFormLoginRequiredLocation
- AuthFormLoginSuccessLocation
- AuthFormLogoutLocation
- AuthFormMethod
- AuthFormMimetype
- AuthFormPassword
- AuthFormProvider
- AuthFormSitePassphrase
- AuthFormSize
- AuthFormUsername
- AuthGroupFile
- AuthLDAPAuthorizePrefix
- AuthLDAPBindAuthoritative
- AuthLDAPBindDN
- AuthLDAPBindPassword
- AuthLDAPCharsetConfig
- AuthLDAPCompareAsUser
- AuthLDAPCompareDNOnServer
- AuthLDAPDereferenceAliases
- AuthLDAPGroupAttribute
- AuthLDAPGroupAttributeIsDN
- AuthLDAPInitialBindAsUser
- AuthLDAPInitialBindPattern
- AuthLDAPMaxSubGroupDepth
- AuthLDAPRemoteUserAttribute
- AuthLDAPRemoteUserIsDN
- AuthLDAPSearchAsUser
- AuthLDAPSubGroupAttribute
- AuthLDAPSubGroupClass
- AuthLDAPUrl
- AuthMerging
- AuthName
- AuthnCacheContext
- AuthnCacheEnable
- AuthnCacheProvideFor
- AuthnCacheSOCache
- AuthnCacheTimeout
- <AuthnProviderAlias>
- AuthnzFcgiCheckAuthnProvider
- AuthnzFcgiDefineProvider
- AuthType
- AuthUserFile
- AuthzDBDLoginToReferer
- AuthzDBDQuery
- AuthzDBDRedirectQuery
- AuthzDBMType
- <AuthzProviderAlias>
- AuthzSendForbiddenOnFailure
- BalancerGrowth
- BalancerInherit
- BalancerMember
- BalancerPersist
- BrotliAlterETag
- BrotliCompressionMaxInputBlock
- BrotliCompressionQuality
- BrotliCompressionWindow
- BrotliFilterNote
- BrowserMatch
- BrowserMatchNoCase
- BufferedLogs
- BufferSize
- CacheDefaultExpire
- CacheDetailHeader
- CacheDirLength
- CacheDirLevels
- CacheDisable
- CacheEnable
- CacheFile
- CacheHeader
- CacheIgnoreCacheControl
- CacheIgnoreHeaders
- CacheIgnoreNoLastMod
- CheckSpelling
- ChrootDir
- ContentDigest
- CookieDomain
- CookieExpires
- CookieName
- CookieStyle
- CookieTracking
- CoreDumpDirectory
- CustomLog
- Dav
- DavDepthInfinity
- DavGenericLockDB
- DavLockDB
- DavMinTimeout
- DBDExptime
- DBDInitSQL
- DBDKeep
- DBDMax
- DBDMin
- DBDParams
- DBDPersist
- DBDPrepareSQL
- DBDriver
- DefaultIcon
- DefaultLanguage
- DefaultRuntimeDir
- DefaultType
- Define
- DeflateBufferSize
- DeflateCompressionLevel
- DeflateFilterNote
- DeflateInflateLimitRequestBody
- DeflateInflateRatioBurst
- `DeflateInflateRatioLimit`
- `DeflateMemLevel`
- `DeflateWindowSize`
- `Deny`
- `<Directory>`
- `DirectoryCheckHandler`
- `DirectoryIndex`
- `DirectoryIndexRedirect`
- `<DirectoryMatch>`
- `DirectorySlash`
- `DocumentRoot`
- `DTracePrivileges`
- `DumpIOInput`
- `DumpIOOutput`
- `<Else>`
- `<ElseIf>`
- `EnableExceptionHook`
- `EnableMMAP`
- `EnableSendfile`
- `Error`
- `ErrorDocument`
- `ErrorLog`
- `ErrorLogFormat`
- `Example`
- `ExpiresActive`
- `ExpiresByType`
- `ExpiresDefault`
- `ExtendedStatus`
- `ExtFilterDefine`
- `ExtFilterOptions`
- `FallbackResource`
- `FileETag`
- `<Files>`
- `<FilesMatch>`
- FilterChain
- FilterDeclare
- FilterProtocol
- FilterProvider
- FilterTrace
- ForceLanguagePriority
- ForceType
- ForensicLog
- GlobalLog
- GprofDir
- GracefulShutdownTimeout
- Group
- H2CopyFiles
- H2Direct
- H2EarlyHints
- H2MaxSessionStreams
- H2MaxWorkerIdleSeconds
- H2MaxWorkers
- H2MinWorkers
- H2ModernTLSOnly
- H2Push
- H2PushDiarySize
- H2PushPriority
- H2PushResource
- H2SerializeHeaders
- H2StreamMaxMemSize
- H2TLSCoolDownSecs
- H2TLSWarmUpSize
- H2Upgrade
- H2WindowSize
- Header
- HeaderName
- HeartbeatAddress
- HeartbeatListen
HeartbeatMaxServers
HeartbeatStorage
HeartbeatStorage
HostnameLookups
HttpProtocolOptions
IdentityCheck
IdentityCheckTimeout

<If>
<IfDefine>
<IfModule>
<IfVersion>
ImapBase
ImapDefault
ImapMenu
Include
IncludeOptional
IndexHeadInsert
IndexIgnore
IndexIgnoreReset
IndexOptions
IndexOrderDefault
IndexStyleSheet
InputSed
ISAPIAppendLogToErrors
ISAPIAppendLogToQuery
ISAPICacheFile
ISAPIFakeAsync
ISAPILogNotSupported
ISAPIReadAheadBuffer
KeepAlive
KeepAliveTimeout
KeptBodySize
LanguagePriority
LDAPCacheEntries
- LDAPCacheTTL
- LDAPConnectionPoolTTL
- LDAPConnectionTimeout
- LDAPLibraryDebug
- LDAPOpCacheEntries
- LDAPOpCacheTTL
- LDAPReferralHopLimit
- LDAPReferrals
- LDAPRetries
- LDAPRetryDelay
- LDAPSharedCacheFile
- LDAPSharedCacheSize
- LDAPTimeout
- LDAPTrustedClientCert
- LDAPTrustedGlobalCert
- LDAPTrustedMode
- LDAPVerifyServerCert
- <Limit>
- <LimitExcept>
- LimitInternalRecursion
- LimitRequestBody
- LimitRequestFields
- LimitRequestFieldSize
- LimitRequestLine
- LimitXMLRequestBody
- Listen
- ListenBackLog
- ListenCoresBucketsRatio
- LoadFile
- LoadModule
- <Location>
- <LocationMatch>
- LogFormat
- LogIOTrackTTFB
- MergeTrailers
- MetaDir
- MetaFiles
- MetaSuffix
- MimeMagicFile
- MinSpareServers
- MinSpareThreads
- MMapFile
- ModemStandard
- ModMimeUsePathInfo
- MultiviewsMatch
- Mutex
- NameVirtualHost
- NoProxy
- NWSSLTrustedCerts
- NWSSLUpgradeable
- Options
- Order
- OutputSed
- PassEnv
- PidFile
- PrivilegesMode
- Protocol
- ProtocolEcho
- Protocols
- ProtocolsHonorOrder
- <Proxy>
- ProxyAddHeaders
- ProxyBadHeader
- ProxyBlock
- ProxyDomain
- ProxyErrorOverride
- ProxyExpressDBMFile
- ProxyExpressDBMType
- ProxyExpressEnable
- ProxyFCGIBackendType
- ProxyFCGISetEnvIf
- ProxyFtpDirCharset
- ProxyFtpEscapeWildcards
- ProxyFtpListOnWildcard
- ProxyHCExpr
- ProxyHCTemplate
- ProxyHCTPsize
- ProxyHTMLBufSize
- ProxyHTMLCharsetOut
- ProxyHTMLDocType
- ProxyHTMLEnable
- ProxyHTMLEvents
- ProxyHTMLExtended
- ProxyHTMLFixups
- ProxyHTMLInterp
- ProxyHTMLLinks
- ProxyHTMLMeta
- ProxyHTMLStripComments
- ProxyHTMLURLMap
- ProxyIOBufferSize
- <ProxyMatch>
- ProxyMaxForwards
- ProxyPass
- ProxyPassInherit
- ProxyPassInterpolateEnv
- ProxyPassMatch
- ProxyPassReverse
- ProxyPassReverseCookieDomain
- ProxyPassReverseCookiePath
- ProxyPreserveHost
- ProxyReceiveBufferSize
- ProxyRemote
- ProxyRemoteMatch
- ProxyRequests
- ProxySCGISInternalRedirect
- ProxySCGISendfile
- ProxySet
- ProxySourceAddress
- ProxyStatus
- ProxyTimeout
- ProxyVia
- QualifyRedirectURL
- README
- ReceiveBufferSize
- Redirect
- RedirectMatch
- RedirectPermanent
- RedirectTemp
- ReflectorHeader
- RegisterHttpMethod
- RemoteIPHeader
- RemoteIPInternalProxy
- RemoteIPInternalProxyList
- RemoteIPProxiesHeader
- RemoteIPTaggedProxy
- RemoteIPTaggedProxyList
- RemoveCharset
- RemoveEncoding
- RemoveHandler
- RemoveInputFilter
- RemoveLanguage
- RemoveOutputFilter
- RemoveType
- RequestHeader
- RequestReadTimeout
- Require
- <RequireAll>
- <RequireAny>
- <RequireNone>
- RewriteBase
- RewriteCond
- RewriteEngine
- RewriteMap
- RewriteOptions
- RewriteRule
- RLimitCPU
- RLimitMEM
- RLimitNPROC
- Satisfy
- ScoreBoardFile
- Script
- ScriptAlias
- ScriptAliasMatch
- ScriptInterpreterSource
- ScriptLog
- ScriptLogBuffer
- ScriptLogLength
- ScriptSock
- SecureListen
- SeeRequestTail
- SendBufferSize
- ServerAdmin
- ServerAlias
- ServerLimit
- ServerName
- ServerPath
- ServerRoot
- ServerSignature
- ServerTokens
- Session
- SessionCookieName
- SessionCookieName2
- SessionCookieRemove
- SessionCryptoCipher
- SessionCryptoDriver
- SessionCryptoPassphrase
- SessionCryptoPassphraseFile
- SessionDBDCookieName
- SessionDBDCookieName2
- SessionDBDCookieRemove
- SessionDBDDDeleteLabel
- SessionDBDInsertLabel
- SessionDBDPerUser
- SessionDBDSelectLabel
- SessionDBDUpdateLabel
- SessionEnv
- SessionExclude
- SessionHeader
- SessionInclude
- SessionMaxAge
- SetEnv
- SetEnvIf
- SetEnvIfExpr
- SetEnvIfNoCase
- SetHandler
- SetInputFilter
- SetOutputFilter
- SSIEndTag
- SSIErrorMsg
- SSIEtTag
- SSILastModified
- SSILegacyExprParser
- SSISrartTag
- SSITimeFormat
- SSIUndefinedEcho
- SSLCACertificateFile
- SSLCACertificatePath
- SSLCADNRequestFile
- SSLCADNRequestPath
- SSLCARevocationCheck
- SSLCARevocationFile
- SSLCARevocationPath
- SSLCertificateChainFile
- SSLCertificateFile
- SSLCertificateKeyFile
- SSLCipherSuite
- SSLCompression
- SSLCryptoDevice
- SSLEngine
- SSLFIPS
- SSLHonorCipherOrder
- SSLInsecureRenegotiation
- SSLOCSPDefaultResponder
- SSLOCSPEnable
- SSLOCSPNoverify
- SSLOCSPOverrideResponder
- SSLOCSPProxyURL
- SSLOCSPResponderCertificateFile
- SSLOCSPResponderTimeout
- SSLOCSPResponseMaxAge
- SSLOCSPResponseTimeSkew
- SSLOCSPUseRequestNonce
- SSLOpenSSLConfCmd
- SSLOptions
- SSLPassPhraseDialog
- SSLProtocol
- SSLProxyCACertificateFile
- SSLProxyCACertificatePath
- SSLProxyCARevocationCheck
- SSLProxyCARevocationFile
- SSLProxyCARevocationPath
- SSLProxyCheckPeerCN
- SSLProxyCheckPeerExpire
- SSLProxyCheckPeerName
- SSLProxyCipherSuite
- SSLProxyEngine
- SSLProxyMachineCertificateChainFile
- SSLProxyMachineCertificateFile
- SSLProxyMachineCertificatePath
- SSLProxyProtocol
- SSLProxyVerify
- SSLProxyVerifyDepth
- SSLRandomSeed
- SSLRenegBufferSize
- SSLRequire
- SSLRequireSSL
- SSLSessionCache
- SSLSessionCacheTimeout
- SSLSessionTicketKeyFile
- SSLSessionTickets
- SSLSRPUnknownUserSeed
- SSLSRPVerifierFile
- SSLStaplingCache
- SSLStaplingErrorCacheTimeout
- SSLStaplingFakeTryLater
- SSLStaplingForceURL
- SSLStaplingResponderTimeout
- SSLStaplingResponseMaxAge
- SSLStaplingResponseTimeSkew
- SSLStaplingReturnResponderErrors
- SSLStaplingStandardCacheTimeout
- SSLStrictSNIVHostCheck
- SSLUserName
- SSLUseStapling
- SSLVerifyClient
- SSLVerifyDepth
- StartServers
- StartThreads
- Substitute
- SubstituteInheritBefore
- SubstituteMaxLineLength
- Suexec
- SuexecUserGroup
- ThreadLimit
- ThreadsPerChild
- ThreadStackSize
- TimeOut
- TraceEnable
- TransferLog
- TypesConfig
- UnDefine
- UndefMacro
- UnsetEnv
- Use
- UseCanonicalName
- UseCanonicalPhysicalPort
- User
- UserDir
- VHostCGIMode
- VHostCGIPrivs
- VHostGroup
- VHostPrivs
- VHostSecure
- VHostUser
- VirtualDocumentRoot
- VirtualDocumentRootIP
- `<VirtualHost>
- VirtualScriptAlias
- VirtualScriptAliasIP
- WatchdogInterval
- XBitHack
- xml2EncAlias
- xml2EncDefault
- xml2StartParse
Apache

<table>
<thead>
<tr>
<th>Feature</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>AcceptFilter protocol accept_filter</td>
<td></td>
</tr>
<tr>
<td>Listen</td>
<td></td>
</tr>
<tr>
<td>AcceptPathInfo On</td>
<td>Off</td>
</tr>
<tr>
<td>AccessFileName filename [filename] ...</td>
<td>.htaccess</td>
</tr>
<tr>
<td>Action action-type cgi-script [virtual]</td>
<td></td>
</tr>
<tr>
<td>CGI</td>
<td></td>
</tr>
<tr>
<td>AddAlt string file [file] ...</td>
<td></td>
</tr>
<tr>
<td>AddAltByEncoding string MIME-encoding [MIME-encoding] ...</td>
<td></td>
</tr>
<tr>
<td>MIME</td>
<td></td>
</tr>
<tr>
<td>AddAltByType string MIME-type [MIME-type] ...</td>
<td></td>
</tr>
<tr>
<td>MIME</td>
<td></td>
</tr>
<tr>
<td>AddCharset charset extension [extension] ...</td>
<td></td>
</tr>
<tr>
<td>AddDefaultCharset On</td>
<td>Off</td>
</tr>
<tr>
<td>text/plain text/html charset</td>
<td></td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>AddDescription</td>
<td>Adds a description for a file.</td>
</tr>
<tr>
<td>AddEncoding</td>
<td>Adds an encoding for a MIME attachment.</td>
</tr>
<tr>
<td>AddHandler</td>
<td>Adds a handler for a MIME type.</td>
</tr>
<tr>
<td>AddIcon</td>
<td>Adds an icon for a MIME type.</td>
</tr>
<tr>
<td>AddIconByEncoding</td>
<td>Adds an icon for a MIME encoding.</td>
</tr>
<tr>
<td>AddIconByType</td>
<td>Adds an icon for a MIME type.</td>
</tr>
<tr>
<td>AddInputFilter</td>
<td>Adds an input filter for a MIME type.</td>
</tr>
<tr>
<td>AddLanguage</td>
<td>Adds a language for a MIME type.</td>
</tr>
<tr>
<td>AddModuleInfo</td>
<td>Adds module information.</td>
</tr>
<tr>
<td>AddOutputFilter</td>
<td>Adds an output filter for a MIME type.</td>
</tr>
<tr>
<td>AddOutputFilterByType</td>
<td>Adds an output filter by media-type.</td>
</tr>
<tr>
<td>AddType</td>
<td>Adds a MIME type.</td>
</tr>
<tr>
<td>Alias</td>
<td>Adds an alias for a URL.</td>
</tr>
<tr>
<td>AliasMatch</td>
<td>Adds a match for a URL regex.</td>
</tr>
<tr>
<td>Setting</td>
<td>Value</td>
</tr>
<tr>
<td>---------</td>
<td>-------</td>
</tr>
<tr>
<td>Allow from all</td>
<td>host[env=[]]env-variable [host[env=[]]env-variable] ...</td>
</tr>
<tr>
<td>Allow CONNECT</td>
<td>port[-port] [port[-port]] ... 443 563</td>
</tr>
<tr>
<td>Ports that are allowed to CONNECT through the proxy</td>
<td></td>
</tr>
<tr>
<td>AllowEncodedSlashes</td>
<td>On</td>
</tr>
<tr>
<td>Allow Methods</td>
<td>reset</td>
</tr>
<tr>
<td>Restrict access to the listed HTTP methods</td>
<td></td>
</tr>
<tr>
<td>AllowOverride</td>
<td>All</td>
</tr>
<tr>
<td>Individual directives that are allowed in .htaccess files</td>
<td></td>
</tr>
<tr>
<td>AllowOverrideList</td>
<td>None</td>
</tr>
<tr>
<td>Anonymous user</td>
<td>user [user] ... userID</td>
</tr>
<tr>
<td>Anonymous LogEmail</td>
<td>On</td>
</tr>
<tr>
<td>Anonymous_MustGiveEmail</td>
<td>On</td>
</tr>
<tr>
<td>Anonymous_NoUserID</td>
<td>Off</td>
</tr>
<tr>
<td>Anonymous_VerifyEmail</td>
<td>Off</td>
</tr>
<tr>
<td>AsyncRequestWorkerFactor</td>
<td>factor Limit concurrent connections per process</td>
</tr>
<tr>
<td>AuthBasicAuthoritative</td>
<td>On</td>
</tr>
<tr>
<td>AuthBasicFake</td>
<td>off</td>
</tr>
<tr>
<td>AuthBasicProvider</td>
<td>provider-name [provider-name] ... file</td>
</tr>
<tr>
<td>Setting</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>AuthBasicUseDigestAlgorithm</td>
<td>MD5</td>
</tr>
<tr>
<td>Check passwords against the authentication providers as if Digest Authentication was in for Basic Authentication.</td>
<td></td>
</tr>
<tr>
<td>AuthDBDUserPWQuery query</td>
<td></td>
</tr>
<tr>
<td>SQL query to look up a password for a user</td>
<td></td>
</tr>
<tr>
<td>AuthDBDUserRealmQuery query</td>
<td></td>
</tr>
<tr>
<td>SQL query to look up a password hash for a user and realm.</td>
<td></td>
</tr>
<tr>
<td>AuthDBMGroupFile file-path</td>
<td></td>
</tr>
<tr>
<td>Sets the name of the database file containing the list of user groups for authorization</td>
<td></td>
</tr>
<tr>
<td>AuthDBMType default</td>
<td>SDBM</td>
</tr>
<tr>
<td>default</td>
<td></td>
</tr>
<tr>
<td>AuthDBMUserFile file-path</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>AuthDigestAlgorithm MD5</td>
<td>MD5-sess</td>
</tr>
<tr>
<td>Selects the algorithm used to calculate the challenge and response hashes in digest authentication</td>
<td></td>
</tr>
<tr>
<td>AuthDigestDomain URI [URI] ...</td>
<td></td>
</tr>
<tr>
<td>URIs that are in the same protection space for digest authentication</td>
<td></td>
</tr>
<tr>
<td>AuthDigestNonceLifetime seconds</td>
<td>300</td>
</tr>
<tr>
<td>How long the server nonce is valid</td>
<td></td>
</tr>
<tr>
<td>AuthDigestProvider provider-name [provider-name] ...</td>
<td>file</td>
</tr>
<tr>
<td>Sets the authentication provider(s) for this location</td>
<td></td>
</tr>
<tr>
<td>AuthDigestQop none</td>
<td>auth</td>
</tr>
<tr>
<td>Determines the quality-of-protection to use in digest authentication</td>
<td></td>
</tr>
<tr>
<td>AuthDigestShmemSize size</td>
<td>1000</td>
</tr>
<tr>
<td>The amount of shared memory to allocate for keeping track of clients</td>
<td></td>
</tr>
<tr>
<td>AuthFormAuthoritative On</td>
<td>Off</td>
</tr>
<tr>
<td>Sets whether authorization and authentication are passed to lower level modules</td>
<td></td>
</tr>
<tr>
<td>AuthFormBody fieldname</td>
<td></td>
</tr>
<tr>
<td>The name of a form field carrying the body of the request to attempt on successful login</td>
<td></td>
</tr>
<tr>
<td>AuthFormDisableNoStore On</td>
<td>Off</td>
</tr>
<tr>
<td>Disable the CacheControl no-store header on the login page</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| **AuthFormFakeBasicAuth** | On|Off  
Fake a Basic Authentication header  |
| **AuthFormLocation** | *fieldname*  
The name of a form field carrying a URL to redirect to on successful login  |
| **AuthFormLoginRequiredLocation** | *url*  
The URL of the page to be redirected to should login be required  |
| **AuthFormLoginSuccessLocation** | *url*  
The URL of the page to be redirected to should login be successful  |
| **AuthFormLogoutLocation** | *uri*  
The URL to redirect to after a user has logged out  |
| **AuthFormMethod** | *fieldname*  
The name of a form field carrying the method of the request to attempt on successful login  |
| **AuthFormMimetype** | *fieldname*  
The name of a form field carrying the mimetype of the body of the request to attempt on successful login  |
| **AuthFormPassword** | *fieldname*  
The name of a form field carrying the login password  |
| **AuthFormProvider** | *provider-name [provider-name] ...*  
Sets the authentication provider(s) for this location  |
| **AuthFormSitePassphrase** | *secret*  
Bypass authentication checks for high traffic sites  |
| **AuthFormSize** | *size*  
The largest size of the form in bytes that will be parsed for the login details  |
| **AuthFormUsername** | *fieldname*  
The name of a form field carrying the login username  |
| **AuthGroupFile** | *file-path*  |
| **AuthLDAPAuthorizePrefix** | *prefix*  
Specifies the prefix for environment variables set during authorization  |
| **AuthLDAPBindAuthoritative** | *off|on*  
Determines if other authentication providers are used when a user can be mapped to a DN cannot successfully bind with the user's credentials.  |
| **AuthLDAPBindDN** | *distinguished-name*  
Optional DN to use in binding to the LDAP server  |
<p>| <strong>AuthLDAPBindPassword</strong> | <em>password</em>  |</p>
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Password used in conjunction with the bind DN</td>
<td></td>
</tr>
<tr>
<td>AuthLDAPCharsetName</td>
<td>Language to charset conversion configuration file</td>
</tr>
<tr>
<td>AuthLDAPCompareAsUser on/off</td>
<td>Use the authenticated user's credentials to perform authorization comparisons</td>
</tr>
<tr>
<td>AuthLDAPCompareDNOnServer on/off</td>
<td>Use the LDAP server to compare the DNs</td>
</tr>
<tr>
<td>AuthLDAPDereferenceAliases</td>
<td>always</td>
</tr>
<tr>
<td>AuthLDAPGroupAttribute attribute</td>
<td>LDAP attributes used to identify the user members of groups.</td>
</tr>
<tr>
<td>AuthLDAPGroupAttributeIsDN on/off</td>
<td>Use the DN of the client username when checking for group membership</td>
</tr>
<tr>
<td>AuthLDAPInitialBindAsUser on/off</td>
<td>Determines if the server does the initial DN lookup using the basic authentication users' own name instead of anonymously or with hard-coded credentials for the server</td>
</tr>
<tr>
<td>AuthLDAPInitialBindPattern regex substitution</td>
<td>Specifies the transformation of the basic authentication username to be used when binding to the LDAP server to perform a DN lookup</td>
</tr>
<tr>
<td>AuthLDAPMaxSubGroupDepth Number</td>
<td>Specifies the maximum sub-group nesting depth that will be evaluated before the user search is discontinued</td>
</tr>
<tr>
<td>AuthLDAPRemoteUserAttribute uid</td>
<td>Use the value of the attribute returned during the user query to set the REMOTE_USER environment variable</td>
</tr>
<tr>
<td>AuthLDAPRemoteUserIsDN on/off</td>
<td>Use the DN of the client username to set the REMOTE_USER environment variable</td>
</tr>
<tr>
<td>AuthLDAPSearchAsUser on/off</td>
<td>Use the authenticated user's credentials to perform authorization searches</td>
</tr>
<tr>
<td>AuthLDAPSubGroupAttribute attribute</td>
<td>Specifies the attribute labels, one value per directive line, used to distinguish the members of a group that are groups.</td>
</tr>
<tr>
<td>AuthLDAPSubGroupClass LdapObjectClass</td>
<td>Specifies which LDAP objectClass values identify directory objects that are groups during sub-group processing.</td>
</tr>
<tr>
<td>AuthLDAPUrl url [NONE</td>
<td>SSL</td>
</tr>
<tr>
<td>Configuration Setting</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>AuthMerging</strong></td>
<td>Off</td>
</tr>
<tr>
<td><strong>AuthName auth-domain</strong></td>
<td>Authorization realm for use in HTTP authentication</td>
</tr>
<tr>
<td>**AuthnCacheContext directory</td>
<td>server</td>
</tr>
<tr>
<td><strong>AuthnCacheEnable</strong></td>
<td>Enable Authn caching configured anywhere</td>
</tr>
<tr>
<td><strong>AuthnCacheProvideFor authn-provider [...]</strong></td>
<td>Specify which authn provider(s) to cache for</td>
</tr>
<tr>
<td><strong>AuthnCacheSOCache provider-name[:provider-args]</strong></td>
<td>Select socache backend provider to use</td>
</tr>
<tr>
<td><strong>AuthnCacheTimeout timeout (seconds)</strong></td>
<td>Set a timeout for cache entries</td>
</tr>
<tr>
<td><strong>&lt;AuthnProviderAlias baseProvider_Alias&gt; ...&lt;/AuthnProviderAlias&gt;</strong></td>
<td>Enclose a group of directives that represent an extension of a base authentication provider and are referenced by the specified alias</td>
</tr>
<tr>
<td>**AuthnzFcgiCheckAuthnProvider provider-name</td>
<td>None option ...**</td>
</tr>
<tr>
<td><strong>AuthnzFcgiDefineProvider type provider-name backend-address</strong></td>
<td>Defines a FastCGI application as a provider for authentication and/or authorization</td>
</tr>
<tr>
<td>**AuthType None</td>
<td>Basic</td>
</tr>
<tr>
<td><strong>AuthUserFile file-path</strong></td>
<td></td>
</tr>
<tr>
<td>**AuthzDBDLoginToReferer On</td>
<td>Off**</td>
</tr>
<tr>
<td><strong>AuthzDBDQuery query</strong></td>
<td></td>
</tr>
<tr>
<td>Specify the SQL Query for the required operation</td>
<td></td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td><strong>AuthzDBDDRedirectQuery</strong> <em>query</em></td>
<td></td>
</tr>
<tr>
<td>Specifying a query to look up a login page for the user</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>AuthzDBMType</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
</tr>
<tr>
<td>Sets the type of database file that is used to store list of user groups</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>&lt;AuthzProviderAlias baseProvider Alias</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Require-Parameters&gt; ...</strong></td>
</tr>
<tr>
<td><strong>&lt;/AuthzProviderAlias&gt;</strong></td>
</tr>
<tr>
<td>Enclose a group of directives that represent an extension of a base authorization provider and the specified alias</td>
</tr>
</tbody>
</table>

| **AuthzSendForbiddenOnFailure On|Off** |
|-------------------------------------|
| Send ‘403 FORBIDDEN’ instead of ‘401 UNAUTHORIZED’ if authentication succeeds but authorization fails |

<table>
<thead>
<tr>
<th><strong>BalancerGrowth #</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of additional Balancers that can be added Post-configuration</td>
</tr>
</tbody>
</table>

| **BalancerInherit On|Off** |
|----------------------------|
| Inherit ProxyPassed Balancers/Workers from the main server |

| **BalancerPersist On|Off** |
|----------------------------|
| Attempt to persist changes made by the Balancer Manager across restarts. |

| **BrotliAlterETag AddSuffix|NoChange|Remove** |
|--------------------------------------|
| How the outgoing ETag header should be modified during compression |

<table>
<thead>
<tr>
<th><strong>BrotliCompressionMaxInputBlock value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum input block size</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BrotliCompressionQuality value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Compression quality</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BrotliCompressionWindow value</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Broto sliding compression window size</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BrotliFilterNote [type] notename</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Places the compression ratio in a note for logging</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BrowserMatch regex [!]env-variable[=value] [!]env-variable[=value] ...</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP User-Agent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>BrowserMatchNoCase regex [!]env-variable[=value] [!]env-variable[=value] ...</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable/Environment</td>
</tr>
<tr>
<td>----------------------</td>
</tr>
<tr>
<td>HTTP User-Agent</td>
</tr>
<tr>
<td>BufferedLogs On</td>
</tr>
<tr>
<td>BufferSize integer</td>
</tr>
<tr>
<td>CacheDefaultExpire seconds</td>
</tr>
<tr>
<td>CacheDetailHeader on</td>
</tr>
<tr>
<td>CacheDirLength length</td>
</tr>
<tr>
<td>CacheDirLevels levels</td>
</tr>
<tr>
<td>CacheDisable url-string</td>
</tr>
<tr>
<td>CacheEnable cache_type url-string</td>
</tr>
<tr>
<td>CacheFile file-path [file-path] ...</td>
</tr>
<tr>
<td>CacheHeader on</td>
</tr>
<tr>
<td>CacheIgnoreCacheControl On</td>
</tr>
<tr>
<td>CacheIgnoreHeaders header-string [header-string] ...</td>
</tr>
<tr>
<td>CacheIgnoreNoLastMod On</td>
</tr>
<tr>
<td>CacheIgnoreQueryString On</td>
</tr>
<tr>
<td>CacheIgnoreURLSessionIdentifiers identifier [identifier] ...</td>
</tr>
<tr>
<td>CacheKeyBaseURL URL</td>
</tr>
</tbody>
</table>
Override the base URL of reverse proxied cache keys.

<table>
<thead>
<tr>
<th><strong>CacheLastModifiedFactor</strong> float</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheLastModified (expiry)</td>
<td></td>
</tr>
</tbody>
</table>

Enable the thundering herd lock.

<table>
<thead>
<tr>
<th><strong>CacheLock</strong> on/off</th>
<th>off</th>
</tr>
</thead>
</table>

Set the maximum possible age of a cache lock.

<table>
<thead>
<tr>
<th><strong>CacheLockMaxAge</strong> integer</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheLockMaxAge</td>
<td></td>
</tr>
</tbody>
</table>

Set the lock path directory.

<table>
<thead>
<tr>
<th><strong>CacheLockPath</strong> directory</th>
<th>/tmp/mod_cache-lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheLockPath</td>
<td></td>
</tr>
</tbody>
</table>

Set the maximum possible age of a cache lock.

<table>
<thead>
<tr>
<th><strong>CacheMaxExpire</strong> seconds</th>
<th>86400 ()</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheMaxExpire</td>
<td></td>
</tr>
</tbody>
</table>

Set the lock path directory.

<table>
<thead>
<tr>
<th><strong>CacheMaxFileSize</strong> bytes</th>
<th>1000000</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheMaxFileSize</td>
<td></td>
</tr>
</tbody>
</table>

Set the lock path directory.

<table>
<thead>
<tr>
<th><strong>CacheMinFileSize</strong> bytes</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheMinFileSize</td>
<td></td>
</tr>
</tbody>
</table>

Enable the thundering herd lock.

<table>
<thead>
<tr>
<th><strong>CacheNegotiatedDocs</strong> On</th>
<th>Off</th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheNegotiatedDocs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Run the cache from the quick handler.

<table>
<thead>
<tr>
<th><strong>CacheQuickHandler</strong> on</th>
<th>off</th>
<th>on</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheQuickHandler</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The minimum size (in bytes) of the document to read and be cached before sending the data downstream.

<table>
<thead>
<tr>
<th><strong>CacheRoot</strong> directory</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheRoot</td>
<td></td>
</tr>
</tbody>
</table>

The minimum time (in milliseconds) that should elapse while reading before data is sent downstream.

<table>
<thead>
<tr>
<th><strong>CacheSocache type[:args]</strong></th>
<th>The shared object cache implementation to use</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheSocache</td>
<td></td>
</tr>
</tbody>
</table>

The maximum size (in bytes) of an entry to be placed in the cache.

<table>
<thead>
<tr>
<th><strong>CacheSocacheMaxSize</strong> bytes</th>
<th>102400</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheSocacheMaxSize</td>
<td></td>
</tr>
</tbody>
</table>

The maximum time (in seconds) for a document to be placed in the cache.

<table>
<thead>
<tr>
<th><strong>CacheSocacheMaxTime</strong> seconds</th>
<th>86400</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheSocacheMaxTime</td>
<td></td>
</tr>
</tbody>
</table>

The minimum time (in seconds) for a document to be placed in the cache.

<table>
<thead>
<tr>
<th><strong>CacheSocacheMinTime</strong> seconds</th>
<th>600</th>
</tr>
</thead>
<tbody>
<tr>
<td>CacheSocacheMinTime</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------</td>
</tr>
<tr>
<td><strong>CacheSocacheReadSize</strong> bytes</td>
<td>0</td>
</tr>
<tr>
<td>The minimum size (in bytes) of the document to read and be cached before sending the data downstream.</td>
<td></td>
</tr>
<tr>
<td><strong>CacheSocacheReadTime</strong> milliseconds</td>
<td>0</td>
</tr>
<tr>
<td>The minimum time (in milliseconds) that should elapse while reading before data is sent downstream.</td>
<td></td>
</tr>
<tr>
<td><strong>CacheStaleOnError</strong> on</td>
<td>off</td>
</tr>
<tr>
<td>Serve stale content in place of 5xx responses.</td>
<td></td>
</tr>
<tr>
<td><strong>CacheStoreExpired</strong> On</td>
<td>Off</td>
</tr>
<tr>
<td>Attempt to cache responses that the server reports as expired.</td>
<td></td>
</tr>
<tr>
<td><strong>CacheStoreNoStore</strong> On</td>
<td>Off</td>
</tr>
<tr>
<td>no-store</td>
<td></td>
</tr>
<tr>
<td><strong>CacheStorePrivate</strong> On</td>
<td>Off</td>
</tr>
<tr>
<td>private</td>
<td></td>
</tr>
<tr>
<td><strong>CGIDScriptTimeout</strong> time[s</td>
<td>ms]</td>
</tr>
<tr>
<td>The length of time to wait for more output from the CGI program.</td>
<td></td>
</tr>
<tr>
<td><strong>CGIMapExtension</strong> cgi-path .extension</td>
<td></td>
</tr>
<tr>
<td>CGI</td>
<td></td>
</tr>
<tr>
<td><strong>CGIPassAuth</strong> On</td>
<td>Off</td>
</tr>
<tr>
<td>Enables passing HTTP authorization headers to scripts as CGI variables.</td>
<td></td>
</tr>
<tr>
<td><strong>CGIVar</strong> variable rule</td>
<td></td>
</tr>
<tr>
<td>Controls how some CGI variables are set.</td>
<td></td>
</tr>
<tr>
<td><strong>CharsetDefault</strong> charset</td>
<td></td>
</tr>
<tr>
<td>Charset to translate into.</td>
<td></td>
</tr>
<tr>
<td><strong>CharsetOptions</strong> option [option] ...</td>
<td>ImplicitAdd</td>
</tr>
<tr>
<td>Configures charset translation behavior.</td>
<td></td>
</tr>
<tr>
<td><strong>CharsetSourceEnc</strong> charset</td>
<td></td>
</tr>
<tr>
<td>Source charset of files.</td>
<td></td>
</tr>
<tr>
<td><strong>CheckCaseOnly</strong> on</td>
<td>off</td>
</tr>
<tr>
<td>Off</td>
<td></td>
</tr>
<tr>
<td><strong>CheckSpelling</strong> on</td>
<td>off</td>
</tr>
<tr>
<td>Off</td>
<td></td>
</tr>
<tr>
<td><strong>ChrootDir</strong> /path/to/directory</td>
<td></td>
</tr>
<tr>
<td>Directory for apache to run chroot(8) after startup.</td>
<td></td>
</tr>
<tr>
<td><strong>ContentDigest</strong> On</td>
<td>Off</td>
</tr>
<tr>
<td>Off</td>
<td></td>
</tr>
<tr>
<td><strong>CookieDomain</strong> domain</td>
<td></td>
</tr>
<tr>
<td>Content-MD5 HTTP</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>The domain to which the tracking cookie applies</td>
<td></td>
</tr>
<tr>
<td><strong>CookieExpires</strong> <em>expiry-period</em></td>
<td></td>
</tr>
<tr>
<td>Expiry time for the tracking cookie</td>
<td></td>
</tr>
<tr>
<td><strong>CookieName</strong> <em>token</em></td>
<td>Apache</td>
</tr>
<tr>
<td>Name of the tracking cookie</td>
<td></td>
</tr>
<tr>
<td><strong>CookieStyle</strong></td>
<td>Netscape</td>
</tr>
<tr>
<td>*Netscape</td>
<td>Cookie</td>
</tr>
<tr>
<td>Format of the cookie header field</td>
<td></td>
</tr>
<tr>
<td><strong>CookieTracking</strong> *on</td>
<td>off*</td>
</tr>
<tr>
<td>Enables tracking cookie</td>
<td></td>
</tr>
<tr>
<td><strong>CoreDumpDirectory</strong> <em>directory</em></td>
<td></td>
</tr>
<tr>
<td>Apache</td>
<td></td>
</tr>
<tr>
<td><strong>CustomLog</strong> *file</td>
<td>pipe format</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Dav</strong> *On</td>
<td>Off</td>
</tr>
<tr>
<td>WebDAV HTTP</td>
<td></td>
</tr>
<tr>
<td><strong>DavDepthInfinity</strong> *on</td>
<td>off*</td>
</tr>
<tr>
<td>PROPFIND, Depth: Infinity</td>
<td></td>
</tr>
<tr>
<td><strong>DavGenericLockDB</strong> <em>file-path</em></td>
<td>DAV</td>
</tr>
<tr>
<td>DAV</td>
<td></td>
</tr>
<tr>
<td><strong>DavLockDB</strong> <em>file-path</em></td>
<td>DAV</td>
</tr>
<tr>
<td>DAV</td>
<td></td>
</tr>
<tr>
<td><strong>DavMinTimeout</strong> <em>seconds</em></td>
<td>0</td>
</tr>
<tr>
<td>DAV</td>
<td></td>
</tr>
<tr>
<td><strong>DBDExptime</strong> <em>time-in-seconds</em></td>
<td>300</td>
</tr>
<tr>
<td>Keepalive time for idle connections</td>
<td></td>
</tr>
<tr>
<td><strong>DBDInitSQL</strong> <em>&quot;SQL statement&quot;</em></td>
<td></td>
</tr>
<tr>
<td>Execute an SQL statement after connecting to a database</td>
<td></td>
</tr>
<tr>
<td><strong>DBDKeep</strong> <em>number</em></td>
<td>2</td>
</tr>
<tr>
<td>Maximum sustained number of connections</td>
<td></td>
</tr>
<tr>
<td><strong>DBDMax</strong> <em>number</em></td>
<td>10</td>
</tr>
<tr>
<td>Maximum number of connections</td>
<td></td>
</tr>
<tr>
<td><strong>DBDMin</strong> <em>number</em></td>
<td>1</td>
</tr>
<tr>
<td>Minimum number of connections</td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>DBDParams</strong> param1=value1[,param2=value2]**</td>
<td>Parameters for database connection</td>
</tr>
<tr>
<td>**DBDPersist On</td>
<td>Off**</td>
</tr>
<tr>
<td><strong>DBDPrepareSQL &quot;SQL statement&quot; label</strong></td>
<td>Define an SQL prepared statement</td>
</tr>
<tr>
<td><strong>DBDriver name</strong></td>
<td>Specify an SQL driver</td>
</tr>
<tr>
<td><strong>DefaultIcon url-path</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DefaultLanguage MIME-lang</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DefaultRuntimeDir directory-path</strong></td>
<td>Base directory for the server run-time files</td>
</tr>
<tr>
<td>**DefaultType MIME-type</td>
<td>none**</td>
</tr>
<tr>
<td><strong>Define parameter-name</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DeflateBufferSize value</strong></td>
<td></td>
</tr>
<tr>
<td>zlib</td>
<td></td>
</tr>
<tr>
<td><strong>DeflateCompressionLevel value</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DeflateFilterNote [type] notename</strong></td>
<td></td>
</tr>
<tr>
<td><strong>DeflateInflateLimitRequestBody value</strong></td>
<td>Maximum size of inflated request bodies</td>
</tr>
<tr>
<td><strong>DeflateInflateRatioBurst value</strong></td>
<td>Maximum number of times the inflation ratio for request bodies can be crossed</td>
</tr>
<tr>
<td><strong>DeflateInflateRatioLimit value</strong></td>
<td>Maximum inflation ratio for request bodies</td>
</tr>
<tr>
<td><strong>DeflateMemLevel value</strong></td>
<td></td>
</tr>
<tr>
<td>zlib</td>
<td></td>
</tr>
<tr>
<td><strong>DeflateWindowSize value</strong></td>
<td></td>
</tr>
<tr>
<td>Zlib</td>
<td></td>
</tr>
<tr>
<td><strong>DEFAULT_REL_RUN</strong></td>
<td></td>
</tr>
<tr>
<td>+</td>
<td></td>
</tr>
<tr>
<td><strong>text/plain</strong></td>
<td></td>
</tr>
</tbody>
</table>

The table above lists various parameters and their descriptions, along with some default values or configurations.
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deny from all host env=!env-variable</td>
<td></td>
</tr>
<tr>
<td>[host]env=!env-variable</td>
<td>...</td>
</tr>
<tr>
<td>&lt;Directory directory-path&gt; ... &lt;/Directory&gt;</td>
<td></td>
</tr>
<tr>
<td>DirectoryCheckHandler On</td>
<td>Off</td>
</tr>
<tr>
<td>Toggle how this module responds when another handler is configured</td>
<td></td>
</tr>
<tr>
<td>DirectoryIndex local-url [local-url] ...</td>
<td></td>
</tr>
<tr>
<td>index.html</td>
<td></td>
</tr>
<tr>
<td>DirectoryIndexRedirect on</td>
<td>off</td>
</tr>
<tr>
<td>Configures an external redirect for directory indexes.</td>
<td></td>
</tr>
<tr>
<td>&lt;DirectoryMatch regex&gt; ... &lt;/DirectoryMatch&gt;</td>
<td></td>
</tr>
<tr>
<td>DirectorySlash On</td>
<td>Off</td>
</tr>
<tr>
<td>DocumentRoot directory-path</td>
<td>/usr/local/apache/h +</td>
</tr>
<tr>
<td>DTracePrivileges On</td>
<td>Off</td>
</tr>
<tr>
<td>Determines whether the privileges required by dtrace are enabled.</td>
<td></td>
</tr>
<tr>
<td>DumpIOInput On</td>
<td>Off</td>
</tr>
<tr>
<td>DumpIOOutput On</td>
<td>Off</td>
</tr>
<tr>
<td>&lt;Else&gt; ... &lt;/Else&gt;</td>
<td></td>
</tr>
<tr>
<td>Contains directives that apply only if the condition of a previous &lt;If&gt; or &lt;ElseIf&gt; section is not satisfied</td>
<td></td>
</tr>
<tr>
<td>&lt;ElseIf expression&gt; ... &lt;/ElseIf&gt;</td>
<td></td>
</tr>
<tr>
<td>Contains directives that apply only if a condition is satisfied by a request at runtime while the previous &lt;If&gt; or &lt;ElseIf&gt; section is not satisfied</td>
<td></td>
</tr>
<tr>
<td>EnableExceptionHook On</td>
<td>Off</td>
</tr>
<tr>
<td>EnableMMAP On</td>
<td>Off</td>
</tr>
<tr>
<td>EnableSendfile On</td>
<td>Off</td>
</tr>
<tr>
<td>sendfile</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td></td>
</tr>
<tr>
<td><strong>Error message</strong></td>
<td>Abort configuration parsing with a custom error message</td>
</tr>
<tr>
<td><strong>ErrorDocument</strong></td>
<td><strong>error-code document</strong></td>
</tr>
<tr>
<td><strong>ErrorLog</strong></td>
<td>**file-path</td>
</tr>
<tr>
<td>logs/error_log</td>
<td>(Uni +</td>
</tr>
<tr>
<td><strong>ErrorLogFormat</strong></td>
<td>**[connection</td>
</tr>
<tr>
<td>Format specification for error log entries</td>
<td></td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>Demonstration directive to illustrate the Apache module API</td>
</tr>
<tr>
<td><strong>ExpiresActive</strong></td>
<td>**On</td>
</tr>
<tr>
<td>Expires</td>
<td></td>
</tr>
<tr>
<td><strong>ExpiresByType</strong></td>
<td><strong>MIME-type &lt;code&gt;seconds</strong></td>
</tr>
<tr>
<td>MIME Expires</td>
<td></td>
</tr>
<tr>
<td><strong>ExpiresDefault</strong></td>
<td><strong>&lt;code&gt;seconds</strong></td>
</tr>
<tr>
<td><strong>ExtendedStatus</strong></td>
<td>**On</td>
</tr>
<tr>
<td>Off[*]</td>
<td></td>
</tr>
<tr>
<td>Keep track of extended status information for each request</td>
<td></td>
</tr>
<tr>
<td><strong>ExtFilterDefine</strong></td>
<td><strong>filtername parameters</strong></td>
</tr>
<tr>
<td><strong>ExtFilterOptions</strong></td>
<td><strong>option [option] ...</strong></td>
</tr>
<tr>
<td>DebugLevel=0 NoLog</td>
<td></td>
</tr>
<tr>
<td><strong>mod_ext_filter</strong></td>
<td>Define a default URL for requests that don't map to a file</td>
</tr>
<tr>
<td><strong>&lt;Files filename&gt;</strong></td>
<td><strong>...&lt;/Files&gt;</strong></td>
</tr>
<tr>
<td><strong>&lt;FilesMatch regex&gt;</strong></td>
<td><strong>...&lt;/FilesMatch&gt;</strong></td>
</tr>
<tr>
<td><strong>FilterChain</strong></td>
<td><strong>[+-@!]filter-name ...</strong></td>
</tr>
<tr>
<td>Configure the filter chain</td>
<td></td>
</tr>
<tr>
<td><strong>FilterDeclare</strong></td>
<td><strong>filter-name [type]</strong></td>
</tr>
<tr>
<td>Declare a smart filter</td>
<td></td>
</tr>
<tr>
<td>FilterProtocol filter-name [provider-name]</td>
<td></td>
</tr>
<tr>
<td><strong>proto-flags</strong></td>
<td></td>
</tr>
<tr>
<td>Deal with correct HTTP protocol handling</td>
<td></td>
</tr>
<tr>
<td>FilterProvider filter-name provider-name</td>
<td></td>
</tr>
<tr>
<td><strong>expression</strong></td>
<td></td>
</tr>
<tr>
<td>Register a content filter</td>
<td></td>
</tr>
<tr>
<td>FilterTrace filter-name level</td>
<td></td>
</tr>
<tr>
<td><strong>expression</strong></td>
<td></td>
</tr>
<tr>
<td>Get debug/diagnostic information from <strong>mod filter</strong></td>
<td></td>
</tr>
<tr>
<td>ForceLanguagePriority None</td>
<td>Prefer</td>
</tr>
<tr>
<td>Prefer</td>
<td></td>
</tr>
<tr>
<td>ForceType MIME-type</td>
<td>None</td>
</tr>
<tr>
<td>MIME</td>
<td></td>
</tr>
<tr>
<td>ForensicLog filename</td>
<td>pipe</td>
</tr>
<tr>
<td>Forensic</td>
<td></td>
</tr>
<tr>
<td>GlobalLogfile</td>
<td>pipe format</td>
</tr>
<tr>
<td>Sets filename and format of log file</td>
<td></td>
</tr>
<tr>
<td>GprofDir /tmp/gprof/</td>
<td>/tmp/gprof/%</td>
</tr>
<tr>
<td>Directory to write gmon.out profiling data to.</td>
<td></td>
</tr>
<tr>
<td>GracefulShutDownTimeout seconds</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Group unix-group</td>
<td></td>
</tr>
<tr>
<td>Group under which the server will answer requests</td>
<td></td>
</tr>
<tr>
<td>H2CopyFiles on</td>
<td>off</td>
</tr>
<tr>
<td>Determine file handling in responses</td>
<td></td>
</tr>
<tr>
<td>H2Direct on</td>
<td>off</td>
</tr>
<tr>
<td>H2 Direct Protocol Switch</td>
<td></td>
</tr>
<tr>
<td>H2EarlyHints on</td>
<td>off</td>
</tr>
<tr>
<td>Determine sending of 103 status codes</td>
<td></td>
</tr>
<tr>
<td>H2MaxSessionStreams n</td>
<td></td>
</tr>
<tr>
<td>Maximum number of active streams per HTTP/2 session.</td>
<td></td>
</tr>
<tr>
<td>H2MaxWorkerIdleSeconds n</td>
<td></td>
</tr>
<tr>
<td>Maximum number of seconds h2 workers remain idle until shut down.</td>
<td></td>
</tr>
<tr>
<td>H2MaxWorkers n</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Max number of worker threads to use per child process.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>H2MinWorkers</strong></td>
<td>n</td>
</tr>
<tr>
<td><strong>H2ModernTLSOnly</strong></td>
<td>on</td>
</tr>
<tr>
<td><strong>Require HTTP/2 connections to be “modern TLS” only</strong></td>
<td>on</td>
</tr>
<tr>
<td><strong>H2Push</strong></td>
<td>on</td>
</tr>
<tr>
<td><strong>H2 Server Push Switch</strong></td>
<td></td>
</tr>
<tr>
<td><strong>H2PushDiarySize</strong></td>
<td>256</td>
</tr>
<tr>
<td><strong>H2 Push Diary Size</strong></td>
<td></td>
</tr>
<tr>
<td><strong>H2PushPriority</strong></td>
<td>mime-type</td>
</tr>
<tr>
<td>**[after]before</td>
<td>interleaved] [weight]**</td>
</tr>
<tr>
<td><strong>H2 Server Push Priority</strong></td>
<td></td>
</tr>
<tr>
<td><strong>H2PushResource</strong></td>
<td>[add] path [critical]</td>
</tr>
<tr>
<td><strong>Declares resources for early pushing to the client</strong></td>
<td></td>
</tr>
<tr>
<td><strong>H2SerializeHeaders</strong></td>
<td>on</td>
</tr>
<tr>
<td><strong>Serialize Request/Response Processing Switch</strong></td>
<td>off</td>
</tr>
<tr>
<td><strong>H2StreamMaxMemSize</strong></td>
<td>65536 bytes</td>
</tr>
<tr>
<td><strong>Maximum amount of output data buffered per stream.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>H2TLSCoolDownSecs</strong></td>
<td>1 seconds</td>
</tr>
<tr>
<td><strong>-</strong></td>
<td></td>
</tr>
<tr>
<td><strong>H2TLSWarmUpSize</strong></td>
<td>1048576 amount</td>
</tr>
<tr>
<td><strong>Size of Stream Window for upstream data.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>H2Upgrade</strong></td>
<td>on</td>
</tr>
<tr>
<td><strong>H2 Upgrade Protocol Switch</strong></td>
<td>on for h2c, off for +</td>
</tr>
<tr>
<td><strong>H2WindowSize</strong></td>
<td>65535 bytes</td>
</tr>
<tr>
<td><strong>Size of Stream Window for upstream data.</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Header</strong></td>
<td>[condition] set</td>
</tr>
<tr>
<td><strong>HTTP</strong></td>
<td></td>
</tr>
<tr>
<td><strong>HeaderName</strong></td>
<td>filename</td>
</tr>
<tr>
<td><strong>HeartbeatAddress</strong></td>
<td>addr:port</td>
</tr>
<tr>
<td><strong>Multicast address for heartbeat packets</strong></td>
<td></td>
</tr>
<tr>
<td><strong>HeartbeatListen addr:port</strong></td>
<td></td>
</tr>
<tr>
<td><strong>multicast address to listen for incoming heartbeat requests</strong></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>HeartbeatMaxServers number-of-servers</td>
<td>10</td>
</tr>
<tr>
<td>Specifies the maximum number of servers that will be sending heartbeat requests to this server</td>
<td></td>
</tr>
<tr>
<td>HeartbeatStorage file-path Path to store heartbeat data</td>
<td>logs/hb.dat</td>
</tr>
<tr>
<td>Path to read heartbeat data</td>
<td>logs/hb.dat</td>
</tr>
<tr>
<td>HostnameLookups On</td>
<td>Off</td>
</tr>
<tr>
<td>HostnameLookups</td>
<td>Off</td>
</tr>
<tr>
<td>HttpProtocolOptions [Strict</td>
<td>Unsafe] [RegisteredMethods</td>
</tr>
<tr>
<td>IdentityCheck On</td>
<td>Off RFC 1413</td>
</tr>
<tr>
<td>IdentityCheckTimeout seconds</td>
<td>30</td>
</tr>
<tr>
<td>IdentityCheckTimeout</td>
<td>30</td>
</tr>
<tr>
<td>&lt;If expression&gt; ... &lt;/If&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;IfDefine ![parameter-name] &gt; ... &lt;/IfDefine&gt;</td>
<td></td>
</tr>
<tr>
<td>&lt;IfModule ![module-file</td>
<td>module-identifier&gt; ... &lt;/IfModule&gt;</td>
</tr>
<tr>
<td>&lt;IfVersion ![operator] version&gt; ... &lt;/IfVersion&gt;</td>
<td></td>
</tr>
<tr>
<td>ImapBase map</td>
<td>referer</td>
</tr>
<tr>
<td>ImapDefault error</td>
<td>nocontent</td>
</tr>
<tr>
<td>ImapMenu none</td>
<td>formatted</td>
</tr>
<tr>
<td>Include file-path</td>
<td>directory-path</td>
</tr>
<tr>
<td>Configuration Item</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>IncludeOptional file-path</td>
<td>directory-path</td>
</tr>
<tr>
<td>IndexHeadInsert &quot;markup ...&quot;</td>
<td></td>
</tr>
<tr>
<td>IndexIgnore file [file] ...</td>
<td></td>
</tr>
<tr>
<td>IndexIgnoreReset ON</td>
<td>OFF</td>
</tr>
<tr>
<td>Empties the list of files to hide when listing a directory</td>
<td></td>
</tr>
<tr>
<td>IndexOptions [+</td>
<td>-]option [+</td>
</tr>
<tr>
<td>IndexOrderDefault Ascending</td>
<td>Descending</td>
</tr>
<tr>
<td>Name</td>
<td>Date</td>
</tr>
<tr>
<td>IndexStyleSheet url-path</td>
<td>CSS</td>
</tr>
<tr>
<td>InputSed sed-command</td>
<td></td>
</tr>
<tr>
<td>Sed command to filter request data (typically POST data)</td>
<td></td>
</tr>
<tr>
<td>ISAPIAppendLogToErrors on</td>
<td>off</td>
</tr>
<tr>
<td>Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the error log</td>
<td></td>
</tr>
<tr>
<td>ISAPIAppendLogToQuery on</td>
<td>off</td>
</tr>
<tr>
<td>Record HSE_APPEND_LOG_PARAMETER requests from ISAPI extensions to the query field</td>
<td></td>
</tr>
<tr>
<td>ISAPICacheFile file-path [file-path] ...</td>
<td></td>
</tr>
<tr>
<td>ISAPI .dll files to be loaded at startup</td>
<td></td>
</tr>
<tr>
<td>ISAPIFakeAsync on</td>
<td>off</td>
</tr>
<tr>
<td>Fake asynchronous support for ISAPI callbacks</td>
<td></td>
</tr>
<tr>
<td>ISAPILogNotSupported on</td>
<td>off</td>
</tr>
<tr>
<td>Log unsupported feature requests from ISAPI extensions</td>
<td></td>
</tr>
<tr>
<td>ISAPIReadAheadBuffer size</td>
<td>49152</td>
</tr>
<tr>
<td>Size of the Read Ahead Buffer sent to ISAPI extensions</td>
<td></td>
</tr>
<tr>
<td>KeepAlive On</td>
<td>Off</td>
</tr>
<tr>
<td>HTTP</td>
<td></td>
</tr>
<tr>
<td>KeepAliveTimeout seconds</td>
<td>5</td>
</tr>
<tr>
<td>KeptBodySize maximum size in bytes</td>
<td>0</td>
</tr>
</tbody>
</table>
Keep the request body instead of discarding it up to the specified maximum size, for potential use by filters such as mod_include.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LanguagePriority MIME-lang [MIME-lang] ... variant</td>
<td></td>
</tr>
<tr>
<td>LDAPCacheEntries number</td>
<td>1024</td>
</tr>
<tr>
<td>Maximum number of entries in the primary LDAP cache</td>
<td></td>
</tr>
<tr>
<td>LDAPCacheTTL seconds</td>
<td>600</td>
</tr>
<tr>
<td>Time that cached items remain valid</td>
<td></td>
</tr>
<tr>
<td>LDAPConnectionPoolTTL n</td>
<td>-1</td>
</tr>
<tr>
<td>Discard backend connections that have been sitting in the connection pool too long</td>
<td></td>
</tr>
<tr>
<td>LDAPConnectionTimeout seconds</td>
<td></td>
</tr>
<tr>
<td>Specifies the socket connection timeout in seconds</td>
<td></td>
</tr>
<tr>
<td>LDAPLibraryDebug 7</td>
<td></td>
</tr>
<tr>
<td>Enable debugging in the LDAP SDK</td>
<td></td>
</tr>
<tr>
<td>LDAPOpCacheEntries number</td>
<td>1024</td>
</tr>
<tr>
<td>Number of entries used to cache LDAP compare operations</td>
<td></td>
</tr>
<tr>
<td>LDAPOpCacheTTL seconds</td>
<td>600</td>
</tr>
<tr>
<td>Time that entries in the operation cache remain valid</td>
<td></td>
</tr>
<tr>
<td>LDAPReferralHopLimit number</td>
<td></td>
</tr>
<tr>
<td>The maximum number of referral hops to chase before terminating an LDAP query.</td>
<td></td>
</tr>
<tr>
<td>LDAPReferrals On</td>
<td>Off</td>
</tr>
<tr>
<td>Enable referral chasing during queries to the LDAP server.</td>
<td></td>
</tr>
<tr>
<td>LDAPRetries number-of-retries</td>
<td>3</td>
</tr>
<tr>
<td>Configures the number of LDAP server retries.</td>
<td></td>
</tr>
<tr>
<td>LDAPRetryDelay seconds</td>
<td>0</td>
</tr>
<tr>
<td>Configures the delay between LDAP server retries.</td>
<td></td>
</tr>
<tr>
<td>LDAPSharedCacheFile directory-path/filename</td>
<td></td>
</tr>
<tr>
<td>Sets the shared memory cache file</td>
<td></td>
</tr>
<tr>
<td>LDAPSharedCacheSize bytes</td>
<td>500000</td>
</tr>
<tr>
<td>Size in bytes of the shared-memory cache</td>
<td></td>
</tr>
<tr>
<td>LDAPTimeout seconds</td>
<td>60</td>
</tr>
<tr>
<td>Specifies the timeout for LDAP search and bind operations, in seconds</td>
<td></td>
</tr>
<tr>
<td>LDAPTrustedClientCert type directory-path/filename/nickname [password]</td>
<td></td>
</tr>
<tr>
<td>Enable verification of client certificates.</td>
<td></td>
</tr>
<tr>
<td>Setting</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>Sets the file or database containing global trusted Certificate Authority or global client certificates.</td>
<td></td>
</tr>
<tr>
<td><strong>LDAPTrustedGlobalCert</strong></td>
<td><code>type directory-path/filename [password]</code></td>
</tr>
<tr>
<td>Specifies the file containing or nickname referring to a per connection client certificate. Not all LDAP toolkits support per connection client certificates.</td>
<td></td>
</tr>
<tr>
<td><strong>LDAPTrustedMode</strong></td>
<td><code>type</code></td>
</tr>
<tr>
<td>Specifies the SSL/TLS mode to be used when connecting to an LDAP server.</td>
<td></td>
</tr>
<tr>
<td><strong>LDAPVerifyServerCert</strong></td>
<td>`On</td>
</tr>
<tr>
<td>Force server certificate verification</td>
<td></td>
</tr>
<tr>
<td><strong>&lt;Limit method [method] ... &gt; ... &lt;/Limit&gt;</strong></td>
<td><code>HTTP</code></td>
</tr>
<tr>
<td>&lt;LimitExcept method [method] ... &gt; ... &lt;/LimitExcept&gt;</td>
<td><code>HTTP</code></td>
</tr>
<tr>
<td><strong>LimitInternalRecursion number [number]</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>LimitRequestBody bytes</strong></td>
<td><code>HTTP</code></td>
</tr>
<tr>
<td><strong>LimitRequestFields number</strong></td>
<td><code>HTTP</code></td>
</tr>
<tr>
<td><strong>LimitRequestFieldSize bytes</strong></td>
<td><code>HTTP</code></td>
</tr>
<tr>
<td><strong>LimitRequestLine bytes</strong></td>
<td><code>HTTP</code></td>
</tr>
<tr>
<td><strong>LimitXMLRequestBody bytes</strong></td>
<td><code>XML</code></td>
</tr>
<tr>
<td><strong>Listen [IP-address:]portnumber [protocol]</strong></td>
<td><code>listen IP</code></td>
</tr>
<tr>
<td><strong>ListenBacklog backlog</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ListenCoresBucketsRatio ratio</strong></td>
<td>0 (disabled)</td>
</tr>
<tr>
<td>Ratio between the number of CPU cores (online) and the number of listeners' buckets</td>
<td></td>
</tr>
<tr>
<td><strong>LoadFile filename [filename]</strong></td>
<td></td>
</tr>
<tr>
<td><strong>LoadModule module filename</strong></td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>`&lt;Location URL-path</td>
<td>URL&gt;`</td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td><code>&lt;LocationMatch regex&gt;</code></td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>LogFormat `format</td>
<td>nickname [nickname]`</td>
</tr>
<tr>
<td>LogIOTrackTTFB `ON</td>
<td>OFF`</td>
</tr>
<tr>
<td>Enable tracking of time to first byte (TTFB)</td>
<td></td>
</tr>
<tr>
<td>LogLevel <code>level</code></td>
<td>warn</td>
</tr>
<tr>
<td>LogMessage <code>message [hook=hook] [expr=expression]</code></td>
<td>Log user-defined message to error log</td>
</tr>
<tr>
<td>LuaAuthzProvider <code>provider_name</code></td>
<td></td>
</tr>
<tr>
<td><code>/path/to/lua/script.lua function_name</code></td>
<td>Plug an authorization provider function into <code>mod_authz_core</code></td>
</tr>
<tr>
<td>LuaCodeCache `stat</td>
<td>forever</td>
</tr>
<tr>
<td>Configure the compiled code cache.</td>
<td></td>
</tr>
<tr>
<td>LuaHookAccessChecker <code>/path/to/lua/script.lua</code></td>
<td></td>
</tr>
<tr>
<td>`hook_function_name [early</td>
<td>late]`</td>
</tr>
<tr>
<td>LuaHookAuthChecker <code>/path/to/lua/script.lua</code></td>
<td></td>
</tr>
<tr>
<td>`hook_function_name [early</td>
<td>late]`</td>
</tr>
<tr>
<td>LuaHookCheckUserID <code>/path/to/lua/script.lua</code></td>
<td></td>
</tr>
<tr>
<td>`hook_function_name [early</td>
<td>late]`</td>
</tr>
<tr>
<td>LuaHookFixups <code>/path/to/lua/script.lua</code></td>
<td></td>
</tr>
<tr>
<td><code>hook_function_name</code></td>
<td>Provide a hook for the fixups phase of a request processing</td>
</tr>
<tr>
<td>LuaHookInsertFilter <code>/path/to/lua/script.lua</code></td>
<td></td>
</tr>
<tr>
<td><code>hook_function_name</code></td>
<td>Provide a hook for the insert_filter phase of request processing</td>
</tr>
<tr>
<td>LuaHookLog <code>/path/to/lua/script.lua</code></td>
<td></td>
</tr>
<tr>
<td>log_function_name</td>
<td>Provide a hook for the access log phase of a request processing</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>LuaHookMapToStorage /path/to/lua/script.lua hook_function_name</td>
<td>Provide a hook for the map_to_storage phase of request processing</td>
</tr>
<tr>
<td>LuaHookTranslateName /path/to/lua/script.lua hook_function_name [early</td>
<td>late]</td>
</tr>
<tr>
<td>LuaHookTypeChecker /path/to/lua/script.lua hook_function_name</td>
<td>Provide a hook for the type_checker phase of request processing</td>
</tr>
<tr>
<td>LuaInherit none</td>
<td>parent-first</td>
</tr>
<tr>
<td>Controls how parent configuration sections are merged into children</td>
<td></td>
</tr>
<tr>
<td>LuaInputFilter filter_name /path/to/lua/script.lua function_name</td>
<td>Provide a Lua function for content input filtering</td>
</tr>
<tr>
<td>LuaMapHandler uri-pattern /path/to/lua/script.lua [function-name]</td>
<td>Map a path to a lua handler</td>
</tr>
<tr>
<td>LuaOutputFilter filter_name /path/to/lua/script.lua function_name</td>
<td>Provide a Lua function for content output filtering</td>
</tr>
<tr>
<td>LuaPackageCPath /path/to/include/?.soa</td>
<td>Add a directory to lua's package.cpath</td>
</tr>
<tr>
<td>LuaPackagePath /path/to/include/?.lua</td>
<td>Add a directory to lua's package.path</td>
</tr>
<tr>
<td>LuaQuickHandler /path/to/script.lua hook_function_name</td>
<td>Provide a hook for the quick handler of request processing</td>
</tr>
<tr>
<td>LuaRoot /path/to/a/directory</td>
<td>Specify the base path for resolving relative paths for mod_lua directives</td>
</tr>
<tr>
<td>LuaScope once</td>
<td>request</td>
</tr>
<tr>
<td>[min] [max]</td>
<td>One of once, request, conn, thread -- default is once</td>
</tr>
<tr>
<td>&lt;Macro name [par1 .. parN]&gt; ... &lt;/Macro&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Define a configuration file macro

<table>
<thead>
<tr>
<th>Macro</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MaxConnectionsPerChild</code> number</td>
<td>0</td>
</tr>
<tr>
<td>Limit on the number of connections that an individual child server will handle during its life</td>
<td></td>
</tr>
<tr>
<td><code>MaxKeepAliveRequests</code> number</td>
<td>100</td>
</tr>
<tr>
<td><code>MaxMemFree</code> KBytes free()</td>
<td>0</td>
</tr>
<tr>
<td><code>MaxRangeOverlaps</code> default</td>
<td>unlimited</td>
</tr>
<tr>
<td>Number of overlapping ranges (eg: 100-200, 150-300) allowed before returning the complete resource</td>
<td></td>
</tr>
<tr>
<td><code>MaxRangeReversals</code> default</td>
<td>unlimited</td>
</tr>
<tr>
<td>Number of range reversals (eg: 100-200, 50-70) allowed before returning the complete resource</td>
<td></td>
</tr>
<tr>
<td><code>MaxRanges</code> default</td>
<td>unlimited</td>
</tr>
<tr>
<td>Number of ranges allowed before returning the complete resource</td>
<td></td>
</tr>
<tr>
<td><code>MaxRequestWorkers</code> number</td>
<td></td>
</tr>
<tr>
<td>Maximum number of connections that will be processed simultaneously</td>
<td></td>
</tr>
<tr>
<td><code>MaxSpareServers</code> number</td>
<td>10</td>
</tr>
<tr>
<td><code>MaxSpareThreads</code> number</td>
<td></td>
</tr>
<tr>
<td><code>MaxThreads</code> number</td>
<td>2048</td>
</tr>
<tr>
<td>Set the maximum number of worker threads</td>
<td></td>
</tr>
<tr>
<td><code>MemcacheConnTTL</code> num[units]</td>
<td>15s</td>
</tr>
<tr>
<td>Keepalive time for idle connections</td>
<td></td>
</tr>
<tr>
<td><code>MergeTrailers</code> [on</td>
<td>off]</td>
</tr>
<tr>
<td>Determines whether trailers are merged into headers</td>
<td></td>
</tr>
<tr>
<td><code>MetaDir</code> directory</td>
<td>.web</td>
</tr>
<tr>
<td>Name of the directory to find CERN-style meta information files</td>
<td></td>
</tr>
<tr>
<td><code>MetaFiles</code> on</td>
<td>off</td>
</tr>
<tr>
<td>Activates CERN meta-file processing</td>
<td></td>
</tr>
<tr>
<td><code>MetaSuffix</code> suffix</td>
<td>.meta</td>
</tr>
<tr>
<td>File name suffix for the file containing CERN-style meta information</td>
<td></td>
</tr>
<tr>
<td><code>MimeMagicFile</code> file-path</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>Value</td>
</tr>
<tr>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>Enable MIME-type determination based on file contents using the specified magic file</td>
<td></td>
</tr>
<tr>
<td><strong>MinSpareServers number</strong></td>
<td>5</td>
</tr>
<tr>
<td><strong>MinSpareThreads number</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MMapFile file-path [file-path] ...</strong></td>
<td>Map a list of files into memory at startup time</td>
</tr>
<tr>
<td>**ModemStandard V.21</td>
<td>V.26bis</td>
</tr>
<tr>
<td>**ModMimeUsePathInfo On</td>
<td>Off**</td>
</tr>
<tr>
<td><strong>path_info mod_mime</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MultiviewsMatch</strong></td>
<td>NegotiatedOnly</td>
</tr>
<tr>
<td>**Any</td>
<td>NegotiatedOnly</td>
</tr>
<tr>
<td><strong>MultiViews</strong></td>
<td></td>
</tr>
<tr>
<td>**Mutex mechanism [default</td>
<td>mutex-name] ... [OmitPID]**</td>
</tr>
<tr>
<td><strong>Configures mutex mechanism and lock file directory for all or specified mutexes</strong></td>
<td></td>
</tr>
<tr>
<td><strong>NameVirtualHost addr[:port]</strong></td>
<td>IP</td>
</tr>
<tr>
<td><strong>NoProxy host [host] ...</strong></td>
<td></td>
</tr>
<tr>
<td><strong>NWSSLTrustedCerts filename [filename] ...</strong></td>
<td>List of additional client certificates</td>
</tr>
<tr>
<td><strong>NWSSLUpgradeable [IP-address:]portnumber</strong></td>
<td>Allows a connection to be upgraded to an SSL connection upon request</td>
</tr>
<tr>
<td>**Options [+</td>
<td>-]option [[+</td>
</tr>
<tr>
<td><strong>Order ordering</strong></td>
<td>Deny,Allow</td>
</tr>
<tr>
<td><strong>Allow</strong></td>
<td>Deny</td>
</tr>
<tr>
<td><strong>OutputSed sed-command</strong></td>
<td>Sed command for filtering response content</td>
</tr>
<tr>
<td><strong>PassEnv env-variable [env-variable] ...</strong></td>
<td></td>
</tr>
<tr>
<td><strong>PidFile filename</strong></td>
<td>logs/httpd.pid</td>
</tr>
<tr>
<td>ID</td>
<td>PrivilegesMode</td>
</tr>
<tr>
<td>----</td>
<td>----------------</td>
</tr>
<tr>
<td></td>
<td>Trade off processing speed and efficiency vs security against malicious privileges-aware code.</td>
</tr>
<tr>
<td>Protocol protocol</td>
<td>Protocol for a listening socket</td>
</tr>
<tr>
<td>Protocols protocol ...</td>
<td>Protocols available for a server/virtual host</td>
</tr>
<tr>
<td></td>
<td>Determines if order of Protocols determines precedence during negotiation</td>
</tr>
<tr>
<td></td>
<td>&lt;Proxy wildcard-url&gt; ...&lt;/Proxy&gt;</td>
</tr>
<tr>
<td>ProxyAddHeaders</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td>Add proxy information in X-Forwarded-* headers</td>
</tr>
<tr>
<td>ProxyBadHeader</td>
<td>IsError</td>
</tr>
</tbody>
</table>
| ProxyBlock *|word|host|domain | *
<p>|    | [word|host|domain] |
| ProxyDomain | Domain |
| ProxyErrorOverride | On|Off | Off |
| ProxyExpressDBMFile | &lt;pathname&gt; | Pathname to DBM file. |
|    | ProxyExpressDBMFile | &lt;type&gt; | DBM type of file. |
| ProxyExpressEnable | [on|off] | Enable the module functionality. |
| ProxyFCGIBackendType | FPM|GENERIC | FPM |
|    | Specify the type of backend FastCGI application |
| ProxyFCGISetEnvIf | conditional-expression | environment-variable-name | value-expression |
|    |</p>
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
</table>
| **Allow variables sent to FastCGI servers to be fixed up**<br><strong>ProxyFtpDirCharset character set**<br>Define the character set for proxied FTP listings**<br><strong>ProxyFtpEscapeWildcards [on|off]**<br>Whether wildcards in requested filenames are escaped when sent to the FTP server**<br><strong>ProxyFtpListOnWildcard [on|off]**<br>Whether wildcards in requested filenames trigger a file listing**<br><strong>ProxyHCEExpr name {ap_expr expression}<br>Creates a named condition expression to use to determine health of the backend based on**<br><strong>ProxyHCTemplate name parameter=setting<br>&lt;...&gt;<br>Creates a named template for setting various health check parameters**<br><strong>ProxyHCTemplate <size><br>Sets the total server-wide size of the threadpool used for the health check workers.**<br><strong>ProxyHTMLBufSize bytes**<br>Sets the buffer size increment for buffering inline scripts and stylesheets.**<br><strong>ProxyHTMLCharsetOut Charset | *<br>Specify a charset for mod_proxy_html output.**<br><strong>ProxyHTMLDocType HTML|XHTML [Legacy]<br>**OR**<br>ProxyHTMLDocType fpi [SGML|XML]<br>Sets an HTML or XHTML document type declaration.**<br><strong>ProxyHTMLEnable On|Off<br>Turns the proxy_html filter on or off.**<br><strong>ProxyHTMLEvents attribute [attribute ...]<br>Specify attributes to treat as scripting events.**<br><strong>ProxyHTMLExtended On|Off<br>Determines whether to fix links in inline scripts, stylesheets, and scripting events.**<br><strong>ProxyHTMLFixups [lowercase] [dospath][reset]<br>Fixes for simple HTML errors.**<br><strong>ProxyHTMLInterp On|Off<br>Enables per-request interpolation of mod_proxy_html rules.**<br><strong>ProxyHTMLLinks element attribute [attribute2 ...]
<table>
<thead>
<tr>
<th>Configuration</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProxyHTMLMeta On</td>
<td>Off</td>
</tr>
<tr>
<td>ProxyHTMLStripComments On</td>
<td>Off</td>
</tr>
<tr>
<td>ProxyHTMLURLMap from-pattern to-pattern [flags] [cond]</td>
<td>Defines a rule to rewrite HTML links.</td>
</tr>
<tr>
<td>ProxyIOBufferSize bytes</td>
<td>8192</td>
</tr>
<tr>
<td>ProxyMatch regex &gt; ... &lt;/ProxyMatch&gt;</td>
<td></td>
</tr>
<tr>
<td>ProxyMaxForwards number</td>
<td>10</td>
</tr>
<tr>
<td>ProxyPass [path] ![url [key=value.key=value ...]]</td>
<td>URL</td>
</tr>
<tr>
<td>ProxyPassInherit On</td>
<td>Off</td>
</tr>
<tr>
<td>Enable Environment Variable interpolation in Reverse Proxy configurations</td>
<td></td>
</tr>
<tr>
<td>Maps remote servers into the local server URL-space using regular expressions</td>
<td></td>
</tr>
<tr>
<td>ProxyPassReverse [path] url</td>
<td>HTTP URL</td>
</tr>
<tr>
<td>ProxyPassReverseCookieDomain internal-domain public-domain</td>
<td>Set-Cookie Domain</td>
</tr>
<tr>
<td>ProxyPassReverseCookiePath internal-path public-path</td>
<td>Reverse Set-Cookie Path</td>
</tr>
<tr>
<td>ProxyPreserveHost On</td>
<td>Off</td>
</tr>
<tr>
<td>ProxyReceiveBufferSize bytes</td>
<td>0</td>
</tr>
<tr>
<td>Feature</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>ProxyRemote match remote-server</td>
<td></td>
</tr>
<tr>
<td>ProxyRemoteMatch regex remote-server</td>
<td></td>
</tr>
<tr>
<td>ProxyRequests On</td>
<td>Off</td>
</tr>
<tr>
<td>ProxySCGIInternalRedirect On</td>
<td>Off</td>
</tr>
<tr>
<td>Enable or disable internal redirect responses from the backend</td>
<td></td>
</tr>
<tr>
<td>ProxySCGISendfile On</td>
<td>Off</td>
</tr>
<tr>
<td>Enable evaluation of X-Sendfile pseudo response header</td>
<td></td>
</tr>
<tr>
<td>Set various Proxy balancer or member parameters</td>
<td></td>
</tr>
<tr>
<td>ProxySourceAddress address</td>
<td></td>
</tr>
<tr>
<td>Set local IP address for outgoing proxy connections</td>
<td></td>
</tr>
<tr>
<td>Show Proxy LoadBalancer status in mod_status</td>
<td></td>
</tr>
<tr>
<td>ProxyTimeout seconds</td>
<td>300</td>
</tr>
<tr>
<td>ProxyVia On</td>
<td>Off</td>
</tr>
<tr>
<td>Via HTTP</td>
<td></td>
</tr>
<tr>
<td>QualifyRedirectURL ON</td>
<td>OFF</td>
</tr>
<tr>
<td>Controls whether the REDIRECT_URL environment variable is fully qualified</td>
<td></td>
</tr>
<tr>
<td>ReadmeName filename</td>
<td></td>
</tr>
<tr>
<td>ReceiveBufferSize bytes TCP</td>
<td>0</td>
</tr>
<tr>
<td>Redirect [status] URL-path URL</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>RedirectMatch [status] regex URL</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>RedirectPermanent URL-path URL</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>RedirectTemp URL-path URL</td>
<td></td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td><strong>ReflectorHeader inputheader [outputheader]</strong></td>
<td>Reflect an input header to the output headers</td>
</tr>
<tr>
<td><strong>RegisterHttpMethod method [method [...]]</strong></td>
<td>Register non-standard HTTP methods</td>
</tr>
<tr>
<td><strong>RemoteIPHeader header-field</strong></td>
<td>Declare the header field which should be parsed for useragent IP addresses</td>
</tr>
<tr>
<td>**RemoteIPInternalProxy proxy-ip</td>
<td>proxy-ip/subnet</td>
</tr>
<tr>
<td><strong>RemoteIPInternalProxyList filename</strong></td>
<td>Declare client intranet IP addresses trusted to present the RemoteIPHeader value</td>
</tr>
<tr>
<td><strong>RemoteIPProxiesHeader HeaderFieldName</strong></td>
<td>Declare the header field which will record all intermediate IP addresses</td>
</tr>
<tr>
<td>**RemoteIPTrustedProxy proxy-ip</td>
<td>proxy-ip/subnet</td>
</tr>
<tr>
<td><strong>RemoteIPTrustedProxyList filename</strong></td>
<td>Declare client intranet IP addresses trusted to present the RemoteIPHeader value</td>
</tr>
<tr>
<td><strong>RemoveCharset extension [extension] ...</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RemoveEncoding extension [extension] ...</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RemoveHandler extension [extension] ...</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RemoveInputFilter extension [extension] ...</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RemoveLanguage extension [extension] ...</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RemoveOutputFilter extension [extension] ...</strong></td>
<td></td>
</tr>
<tr>
<td><strong>RemoveType extension [extension] ...</strong></td>
<td></td>
</tr>
<tr>
<td>**RequestHeader set</td>
<td>append</td>
</tr>
<tr>
<td>Directive</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>RequestReadTimeout [header=timeout[-maxtimeout][,MinRate=rate]] [body=timeout[-maxtimeout][,MinRate=rate]]</td>
<td>Set timeout values for receiving request headers and body from client.</td>
</tr>
<tr>
<td>Require [not] entity-name [entity-name] ...</td>
<td>Tests whether an authenticated user is authorized by an authorization provider.</td>
</tr>
<tr>
<td>&lt;RequireAll&gt; ... &lt;/RequireAll&gt;</td>
<td>Enclose a group of authorization directives of which none must fail and at least one must succeed for the enclosing directive to succeed.</td>
</tr>
<tr>
<td>&lt;RequireAny&gt; ... &lt;/RequireAny&gt;</td>
<td>Enclose a group of authorization directives of which one must succeed for the enclosing directive to succeed.</td>
</tr>
<tr>
<td>&lt;RequireNone&gt; ... &lt;/RequireNone&gt;</td>
<td>Enclose a group of authorization directives of which none must succeed for the enclosing directive to not fail.</td>
</tr>
<tr>
<td>RewriteBase URL-path</td>
<td>Sets the base URL for per-directory rewrites.</td>
</tr>
<tr>
<td>RewriteCond TestString CondPattern [flags]</td>
<td>Defines a condition under which rewriting will take place.</td>
</tr>
<tr>
<td>RewriteEngine on</td>
<td>off</td>
</tr>
<tr>
<td>RewriteMap MapName MapType:MapSource</td>
<td>Defines a mapping function for key-lookup</td>
</tr>
<tr>
<td>RewriteOptions Options</td>
<td>Sets some special options for the rewrite engine.</td>
</tr>
<tr>
<td>RLimitCPU seconds</td>
<td>max [seconds]max</td>
</tr>
<tr>
<td>RLimitMEM bytes</td>
<td>max [bytes]max</td>
</tr>
<tr>
<td>RLimitNPROC number</td>
<td>max [number]max</td>
</tr>
<tr>
<td>Satisfy Any</td>
<td>All</td>
</tr>
<tr>
<td>ScoreBoardFile file-path</td>
<td>logs/apache_status</td>
</tr>
<tr>
<td>Setting</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>--------------------------------------------</td>
</tr>
<tr>
<td><strong>Script method</strong></td>
<td><strong>cgi-script</strong></td>
</tr>
<tr>
<td><strong>ScriptAlias</strong></td>
<td>**URL-path file-path</td>
</tr>
<tr>
<td><strong>ScriptAliasMatch</strong></td>
<td>**regex file-path</td>
</tr>
<tr>
<td><strong>ScriptInterpreterSource</strong></td>
<td>**Registry</td>
</tr>
<tr>
<td><strong>ScriptLog</strong></td>
<td><strong>file-path</strong></td>
</tr>
<tr>
<td><strong>ScriptLogBuffer</strong></td>
<td><strong>bytes</strong></td>
</tr>
<tr>
<td><strong>ScriptLogLength</strong></td>
<td><strong>bytes</strong></td>
</tr>
<tr>
<td><strong>ScriptSock</strong></td>
<td><strong>file-path</strong></td>
</tr>
<tr>
<td><strong>SecureListen</strong></td>
<td><strong>[IP-address:]portnumber</strong></td>
</tr>
<tr>
<td><strong>Certificate-Name</strong></td>
<td><strong>[MUTUAL]</strong></td>
</tr>
<tr>
<td><strong>SeeRequestTail</strong></td>
<td>**On</td>
</tr>
<tr>
<td><strong>SendBufferSize</strong></td>
<td><strong>bytes</strong></td>
</tr>
<tr>
<td><strong>ServerAdmin</strong></td>
<td>**email-address</td>
</tr>
<tr>
<td><strong>ServerAlias</strong></td>
<td><strong>hostname [hostname] ...</strong></td>
</tr>
<tr>
<td><strong>ServerLimit</strong></td>
<td><strong>number</strong></td>
</tr>
<tr>
<td><strong>ServerName</strong></td>
<td><strong>[scheme://]fully-qualified-domain-name[:port]</strong></td>
</tr>
<tr>
<td><strong>ServerPath</strong></td>
<td><strong>URL-path</strong></td>
</tr>
<tr>
<td>URL</td>
<td>ServerRoot directory-path</td>
</tr>
<tr>
<td>--------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td></td>
<td>/usr/local/apache</td>
</tr>
</tbody>
</table>

| ServerSignature | On|Off|EMail | Off |

| ServerTokens | Major|Minor|Min[imal]|Prod[uctOnly]|OS|Full | Full |
|             |      |      |      |             |   |     |     |

| Server HTTP | Session On|Off | Enables a session for the current directory or location | Off |
|             |           |     |                                                |     |

| SessionCookieName | name attributes | Name and attributes for the RFC2109 cookie storing the session | |
|                  |                 |                                                     |   |

| SessionCookieName2 | name attributes | Name and attributes for the RFC2965 cookie storing the session | |
|                   |                 |                                                     |   |

| SessionCookieRemove | On|Off | Control for whether session cookies should be removed from incoming HTTP headers | Off |
|                    |    |     |                                               |     |

| SessionCryptoCipher | name | The crypto cipher to be used to encrypt the session | |
|                    |      |                                                     |   |

| SessionCryptoDriver | name [param [=value]] | The crypto driver to be used to encrypt the session | |
|                     |                    |                                                     |   |

| SessionCryptoPassphrase | secret [ secret ... ] | The key used to encrypt the session | |
|                         |                    |                                                     |   |

| SessionCryptoPassphraseFile | filename | File containing keys used to encrypt the session | |
|                            |          |                                                     |   |

| SessionDBDCookieName | name attributes | Name and attributes for the RFC2109 cookie storing the session ID | |
|                     |                 |                                                     |   |

| SessionDBDCookieName2 | name attributes | Name and attributes for the RFC2965 cookie storing the session ID | |
|                      |                 |                                                     |   |

| SessionDBDCookieRemove | On|Off | Control for whether session ID cookies should be removed from incoming HTTP headers | On |
|                        |    |     |                                               |     |

| SessionDBDDeleteLabel | label | The SQL query to use to remove sessions from the database | deletesession |
|                      |       |                                                     |   |

| SessionDBDInsertLabel | label | The SQL query to use to insert sessions into the database | insertsession |
|                      |       |                                                     |   |

<p>| SessionDBDPerUser | On|Off | The SQL query to use to insert sessions into the database | Off |
|                  |    |     |                                               |     |</p>
<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Enable a per user session</strong></td>
<td><strong>SessionDBDSelectLabel</strong> <code>label</code> selectsession&lt;br&gt;The SQL query to use to select sessions from the database</td>
</tr>
<tr>
<td><strong>SessionDBDUpdateLabel</strong> <code>label</code></td>
<td>updatesession&lt;br&gt;The SQL query to use to update existing sessions in the database</td>
</tr>
<tr>
<td>**SessionEnv On</td>
<td>Off**</td>
</tr>
<tr>
<td><strong>SessionExclude <code>path</code></strong></td>
<td>Define URL prefixes for which a session is ignored</td>
</tr>
<tr>
<td><strong>SessionHeader <code>header</code></strong></td>
<td>Import session updates from a given HTTP response header</td>
</tr>
<tr>
<td><strong>SessionInclude <code>path</code></strong></td>
<td>Define URL prefixes for which a session is valid</td>
</tr>
<tr>
<td><strong>SessionMaxAge <code>maxage</code></strong></td>
<td>0&lt;br&gt;Define a maximum age in seconds for a session</td>
</tr>
<tr>
<td><strong>SetEnv <code>env-variable</code> <code>value</code></strong></td>
<td>Sets environment variables based on an <code>ap_expr</code> expression</td>
</tr>
<tr>
<td><strong>SetEnvIf attribute regex [!]env-variable[=value] [!]env-variable[=value] ...</strong></td>
<td>Sets environment variables based on an <code>ap_expr</code> expression</td>
</tr>
<tr>
<td><strong>SetEnvIfNoCase attribute regex [!]env-variable[=value] [!]env-variable[=value] ...</strong></td>
<td>Sets environment variables based on an <code>ap_expr</code> expression</td>
</tr>
<tr>
<td>**SetHandler <code>handler-name</code></td>
<td>None**</td>
</tr>
<tr>
<td><strong>SetInputFilter <code>filter[;filter...]</code></strong></td>
<td>Sets environment variables based on an <code>ap_expr</code> expression</td>
</tr>
<tr>
<td><strong>SetOutputFilter <code>filter[;filter...]</code></strong></td>
<td>Sets environment variables based on an <code>ap_expr</code> expression</td>
</tr>
</tbody>
</table>
| **SSIEndTag `tag`** | "-->"<br>Include<br>**SSIErrorMsg `message`** | "[an error occurred +"
<table>
<thead>
<tr>
<th>Setting</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSIETag</td>
<td>on, off</td>
</tr>
<tr>
<td>Controls whether ETags are generated by the server.</td>
<td></td>
</tr>
<tr>
<td>SSLLastModified</td>
<td>on, off</td>
</tr>
<tr>
<td>Controls whether Last-Modified headers are generated by the server.</td>
<td></td>
</tr>
<tr>
<td>SSLLegacyExprParser</td>
<td>on, off</td>
</tr>
<tr>
<td>Enable compatibility mode for conditional expressions.</td>
<td></td>
</tr>
<tr>
<td>SSISTartTag</td>
<td>tag, include</td>
</tr>
<tr>
<td>&quot;&lt;!--#&quot;</td>
<td></td>
</tr>
<tr>
<td>SSITimeFormat</td>
<td>formatstring</td>
</tr>
<tr>
<td>&quot;%A, %d-%b-%Y %H:%M&quot;</td>
<td></td>
</tr>
<tr>
<td>SSIUndefinedEcho</td>
<td>string, echo</td>
</tr>
<tr>
<td>&quot;(none)&quot;</td>
<td></td>
</tr>
<tr>
<td>SSLCACertificateFile</td>
<td>file-path</td>
</tr>
<tr>
<td>File of concatenated PEM-encoded CA Certificates for Client Auth</td>
<td></td>
</tr>
<tr>
<td>SSLCACertificatePath</td>
<td>directory-path</td>
</tr>
<tr>
<td>Directory of PEM-encoded CA Certificates for Client Auth</td>
<td></td>
</tr>
<tr>
<td>SSLCADNRequestFile</td>
<td>file-path</td>
</tr>
<tr>
<td>File of concatenated PEM-encoded CA Certificates for defining acceptable CA names</td>
<td></td>
</tr>
<tr>
<td>SSLCADNRequestPath</td>
<td>directory-path</td>
</tr>
<tr>
<td>Directory of PEM-encoded CA Certificates for defining acceptable CA names</td>
<td></td>
</tr>
<tr>
<td>SSLCARevocationCheck</td>
<td>chain</td>
</tr>
<tr>
<td>Enable CRL-based revocation checking</td>
<td></td>
</tr>
<tr>
<td>SSLCARevocationFile</td>
<td>file-path</td>
</tr>
<tr>
<td>File of concatenated PEM-encoded CA CRLs for Client Auth</td>
<td></td>
</tr>
<tr>
<td>SSLCARevocationPath</td>
<td>directory-path</td>
</tr>
<tr>
<td>Directory of PEM-encoded CA CRLs for Client Auth</td>
<td></td>
</tr>
<tr>
<td>SSLCertificateChainFile</td>
<td>file-path</td>
</tr>
<tr>
<td>File of PEM-encoded Server CA Certificates</td>
<td></td>
</tr>
<tr>
<td>SSLCertificateFile</td>
<td>file-path</td>
</tr>
<tr>
<td>Server PEM-encoded X.509 certificate data file</td>
<td></td>
</tr>
<tr>
<td>SSLCertificateKeyFile</td>
<td>file-path</td>
</tr>
<tr>
<td>Server PEM-encoded private key file</td>
<td></td>
</tr>
<tr>
<td>SSLCipherSuite</td>
<td>cipher-spec</td>
</tr>
<tr>
<td>Cipher Suite available for negotiation in SSL handshake</td>
<td>DEFAULT (depends on Cipher Suite available for negotiation in SSL handshake)</td>
</tr>
<tr>
<td>**SSLCompression on</td>
<td>off**</td>
</tr>
<tr>
<td>------------------</td>
<td>-----</td>
</tr>
<tr>
<td>Enable compression on the SSL level</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SSLCryptoDevice engine</strong></th>
<th>builtin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable use of a cryptographic hardware accelerator</td>
<td></td>
</tr>
</tbody>
</table>

| **SSLEngine on|off|optional** | off |
|------------------|-----|
| SSL Engine Operation Switch |

| **SSLFIPS on|off** | off |
|------------------|-----|
| SSL FIPS mode Switch |

| **SSLHonorCipherOrder on|off** | off |
|------------------|-----|
| Option to prefer the server's cipher preference order |

| **SSLInsecureRenegotiation on|off** | off |
|------------------|-----|
| Option to enable support for insecure renegotiation |

<table>
<thead>
<tr>
<th><strong>SSLOCSDefaultResponder uri</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Set the default responder URI for OCSP validation</td>
<td></td>
</tr>
</tbody>
</table>

| **SSLOCSPEnable on|off** | off |
|------------------|-----|
| Enable OCSP validation of the client certificate chain |

<table>
<thead>
<tr>
<th><strong>SSLOCSPNoverify On/Off</strong></th>
<th>Off</th>
</tr>
</thead>
<tbody>
<tr>
<td>skip the OCSP responder certificates verification</td>
<td></td>
</tr>
</tbody>
</table>

| **SSLOCSPOverrideResponder on|off** | off |
|------------------|-----|
| Force use of the default responder URI for OCSP validation |

<table>
<thead>
<tr>
<th><strong>SSLOCSPProxyURL url</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Proxy URL to use for OCSP requests</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SSLOCSPResponderCertificateFile file</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Set of trusted PEM encoded OCSP responder certificates</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SSLOCSPResponderTimeout seconds</strong></th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeout for OCSP queries</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SSLOCSPResponseMaxAge seconds</strong></th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum allowable age for OCSP responses</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>SSLOCSPResponseTimeSkew seconds</strong></th>
<th>300</th>
</tr>
</thead>
<tbody>
<tr>
<td>Maximum allowable time skew for OCSP response validation</td>
<td></td>
</tr>
</tbody>
</table>

| **SSLOCSPUseRequestNonce on|off** | on |
|------------------|-----|
| Use a nonce within OCSP queries |

<table>
<thead>
<tr>
<th><strong>SSLOpenSSLConfCmd command-name command-value</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Configure OpenSSL parameters through its SSL_CONF API</td>
<td></td>
</tr>
<tr>
<td><strong>SSLOptions [+-]option ...</strong></td>
<td>Configure various SSL engine run-time options</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td><strong>SSLPassPhraseDialog type</strong></td>
<td><code>builtin</code></td>
</tr>
<tr>
<td>Configure pass phrase dialog for encrypted private keys</td>
<td></td>
</tr>
<tr>
<td><strong>SSLProtocol [+-]protocol ...</strong></td>
<td><code>all -SSLv3 (up to 2 +)</code></td>
</tr>
<tr>
<td>Configure usable SSL/TLS protocol versions</td>
<td></td>
</tr>
<tr>
<td><strong>SSLProxyCACertificateFile file-path</strong></td>
<td>File of concatenated PEM-encoded CA Certificates for Remote Server Auth</td>
</tr>
<tr>
<td><strong>SSLProxyCACertificatePath directory-path</strong></td>
<td>Directory of PEM-encoded CA Certificates for Remote Server Auth</td>
</tr>
<tr>
<td>**SSLProxyCARevocationCheck chain</td>
<td>leaf</td>
</tr>
<tr>
<td>Enable CRL-based revocation checking for Remote Server Auth</td>
<td></td>
</tr>
<tr>
<td><strong>SSLProxyCARevocationFile file-path</strong></td>
<td>File of concatenated PEM-encoded CA CRLs for Remote Server Auth</td>
</tr>
<tr>
<td><strong>SSLProxyCARevocationPath directory-path</strong></td>
<td>Directory of PEM-encoded CA CRLs for Remote Server Auth</td>
</tr>
<tr>
<td>**SSLProxyCheckPeerCN on</td>
<td>off**</td>
</tr>
<tr>
<td>Whether to check the remote server certificate's CN field</td>
<td></td>
</tr>
<tr>
<td>**SSLProxyCheckPeerExpire on</td>
<td>off**</td>
</tr>
<tr>
<td>Whether to check if remote server certificate is expired</td>
<td></td>
</tr>
<tr>
<td>**SSLProxyCheckPeerName on</td>
<td>off**</td>
</tr>
<tr>
<td>Configure host name checking for remote server certificates</td>
<td></td>
</tr>
<tr>
<td><strong>SSLProxyCipherSuite cipher-spec</strong></td>
<td><code>ALL:!ADH:RC4+RSA:</code></td>
</tr>
<tr>
<td>Cipher Suite available for negotiation in SSL proxy handshake</td>
<td></td>
</tr>
<tr>
<td>**SSLProxyEngine on</td>
<td>off**</td>
</tr>
<tr>
<td>SSL Proxy Engine Operation Switch</td>
<td></td>
</tr>
<tr>
<td><strong>SSLProxyMachineCertificateChainFile filename</strong></td>
<td>File of concatenated PEM-encoded CA certificates to be used by the proxy for choosing a certificate</td>
</tr>
<tr>
<td><strong>SSLProxyMachineCertificateFile filename</strong></td>
<td>File of concatenated PEM-encoded client certificates and keys to be used by the proxy</td>
</tr>
<tr>
<td><strong>SSLProxyMachineCertificatePath directory</strong></td>
<td>Directory of PEM-encoded client certificates and keys to be used by the proxy</td>
</tr>
<tr>
<td><strong>SSLProxyProtocol [+-]protocol ...</strong></td>
<td><code>all -SSLv3 (up to 2 +)</code></td>
</tr>
<tr>
<td>Configuration Parameter</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------------------------------------</td>
<td>------------------------</td>
</tr>
<tr>
<td><strong>Configure usable SSL protocol flavors for proxy usage</strong></td>
<td></td>
</tr>
<tr>
<td><strong>SSLProxyVerify level</strong></td>
<td><strong>none</strong></td>
</tr>
<tr>
<td>Type of remote server Certificate verification</td>
<td></td>
</tr>
<tr>
<td><strong>SSLProxyVerifyDepth number</strong></td>
<td>1</td>
</tr>
<tr>
<td>Maximum depth of CA Certificates in Remote Server Certificate verification</td>
<td></td>
</tr>
<tr>
<td><strong>SSLRandomSeed context source [bytes]</strong></td>
<td></td>
</tr>
<tr>
<td>Pseudo Random Number Generator (PRNG) seeding source</td>
<td></td>
</tr>
<tr>
<td><strong>SSLRenegBufferSize bytes</strong></td>
<td>131072</td>
</tr>
<tr>
<td>Set the size for the SSL renegotiation buffer</td>
<td></td>
</tr>
<tr>
<td><strong>SSLRequire expression</strong></td>
<td></td>
</tr>
<tr>
<td>Allow access only when an arbitrarily complex boolean expression is true</td>
<td></td>
</tr>
<tr>
<td><strong>SSLRequireSSL</strong></td>
<td></td>
</tr>
<tr>
<td>Deny access when SSL is not used for the HTTP request</td>
<td></td>
</tr>
<tr>
<td><strong>SSLSessionCache type</strong></td>
<td><strong>none</strong></td>
</tr>
<tr>
<td>Type of the global/inter-process SSL Session Cache</td>
<td></td>
</tr>
<tr>
<td><strong>SSLSessionCacheTimeout seconds</strong></td>
<td>300</td>
</tr>
<tr>
<td>Number of seconds before an SSL session expires in the Session Cache</td>
<td></td>
</tr>
<tr>
<td><strong>SSLSessionTicketKeyFile file-path</strong></td>
<td></td>
</tr>
<tr>
<td>Persistent encryption/decryption key for TLS session tickets</td>
<td></td>
</tr>
<tr>
<td>**SSLSessionTickets on</td>
<td>off**</td>
</tr>
<tr>
<td>Enable or disable use of TLS session tickets</td>
<td></td>
</tr>
<tr>
<td><strong>SSLSRPUnknownUserSeed secret-string</strong></td>
<td></td>
</tr>
<tr>
<td>SRP unknown user seed</td>
<td></td>
</tr>
<tr>
<td><strong>SSLSRPVerifierFile file-path</strong></td>
<td></td>
</tr>
<tr>
<td>Path to SRP verifier file</td>
<td></td>
</tr>
<tr>
<td><strong>SSLStaplingCache type</strong></td>
<td></td>
</tr>
<tr>
<td>Configures the OCSP stapling cache</td>
<td></td>
</tr>
<tr>
<td><strong>SSLStaplingErrorCacheTimeout seconds</strong></td>
<td>600</td>
</tr>
<tr>
<td>Number of seconds before expiring invalid responses in the OCSP stapling cache</td>
<td></td>
</tr>
<tr>
<td>**SSLStaplingFakeTryLater on</td>
<td>off**</td>
</tr>
<tr>
<td>Synthesize &quot;tryLater&quot; responses for failed OCSP stapling queries</td>
<td></td>
</tr>
<tr>
<td><strong>SSLStaplingForceURL uri</strong></td>
<td></td>
</tr>
<tr>
<td>Override the OCSP responder URI specified in the certificate's AIA extension</td>
<td></td>
</tr>
<tr>
<td><strong>SSLStaplingResponderTimeout seconds</strong></td>
<td>10</td>
</tr>
<tr>
<td>Timeout for OCSP stapling queries</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>SSLStaplingResponseMaxAge seconds</td>
<td>-1</td>
</tr>
<tr>
<td>SSLStaplingResponseTimeSkew seconds</td>
<td>300</td>
</tr>
<tr>
<td>SSLStaplingReturnResponderErrors on</td>
<td>on</td>
</tr>
<tr>
<td>SSLStaplingStandardCacheTimeout seconds</td>
<td>3600</td>
</tr>
<tr>
<td>SSLStrictSNIVHostCheck on</td>
<td>off</td>
</tr>
<tr>
<td>SSLUserName varname</td>
<td></td>
</tr>
<tr>
<td>SSLUseStapling on</td>
<td>off</td>
</tr>
<tr>
<td>SSLVerifyClient level</td>
<td>none</td>
</tr>
<tr>
<td>SSLVerifyDepth number</td>
<td>1</td>
</tr>
<tr>
<td>StartServers number</td>
<td></td>
</tr>
<tr>
<td>StartThreads number</td>
<td></td>
</tr>
<tr>
<td>Substitute s/pattern/substitution/[infg]</td>
<td></td>
</tr>
<tr>
<td>SubstituteInheritBefore on</td>
<td>off</td>
</tr>
<tr>
<td>SubstituteMaxLineLength bytes(b</td>
<td>B</td>
</tr>
<tr>
<td>Suexec On</td>
<td>Off</td>
</tr>
<tr>
<td>SuexecUserGroup User Group CGI</td>
<td></td>
</tr>
<tr>
<td>ThreadLimit number</td>
<td></td>
</tr>
<tr>
<td>Configuration Item</td>
<td>Setting</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td>ThreadsPerChild number</td>
<td></td>
</tr>
<tr>
<td>ThreadStackSize size</td>
<td></td>
</tr>
<tr>
<td>TimeOut seconds</td>
<td>60</td>
</tr>
<tr>
<td>TraceEnable [on</td>
<td>off</td>
</tr>
<tr>
<td>TransferLog file</td>
<td>pipe</td>
</tr>
<tr>
<td>TypesConfig file-path</td>
<td>conf/mime.types</td>
</tr>
<tr>
<td>UnDefine parameter-name</td>
<td></td>
</tr>
<tr>
<td>UndefMacro name</td>
<td></td>
</tr>
<tr>
<td>UnsetEnv env-variable [env-variable]</td>
<td>...</td>
</tr>
<tr>
<td>Use name [value1 ... valueN]</td>
<td></td>
</tr>
<tr>
<td>UseCanonicalName On</td>
<td>Off</td>
</tr>
<tr>
<td>UseCanonicalPhysicalPort On</td>
<td>Off</td>
</tr>
<tr>
<td>User unix-userid</td>
<td># -1</td>
</tr>
<tr>
<td>UserDir directory-filename [directory-filename]</td>
<td>...</td>
</tr>
<tr>
<td>VHostCGIMode On</td>
<td>Off</td>
</tr>
<tr>
<td>VHostPrivs [+-]?privilege-name [+-]?privilege-name</td>
<td>...</td>
</tr>
<tr>
<td>VHostGroup unix-groupid</td>
<td></td>
</tr>
<tr>
<td>Configuration</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>VHostPrivs</strong></td>
<td>Sets the Group ID under which a virtual host runs.</td>
</tr>
<tr>
<td><code>VHostPrivs [+-]?privilege-name</code></td>
<td>Assign arbitrary privileges to a virtual host.</td>
</tr>
<tr>
<td><strong>VHostSecure</strong></td>
<td>Determines whether the server runs with enhanced security for the virtual host. On</td>
</tr>
<tr>
<td><strong>VHostUser</strong></td>
<td>Sets the User ID under which a virtual host runs.</td>
</tr>
<tr>
<td><strong>VirtualDocumentRoot</strong></td>
<td>Interpolated directory</td>
</tr>
<tr>
<td><strong>VirtualDocumentRootIP</strong></td>
<td>Interpolated directory</td>
</tr>
<tr>
<td><strong>VirtualScriptAlias</strong></td>
<td>Interpolated directory</td>
</tr>
<tr>
<td><strong>VirtualScriptAliasIP</strong></td>
<td>Interpolated directory</td>
</tr>
<tr>
<td><strong>WatchdogInterval</strong></td>
<td>Number-of-seconds</td>
</tr>
<tr>
<td><strong>XBitHack</strong></td>
<td>On/off/full</td>
</tr>
<tr>
<td><strong>xml2EncAlias</strong></td>
<td>CharSet alias</td>
</tr>
<tr>
<td><strong>xml2EncDefault</strong></td>
<td>Name</td>
</tr>
<tr>
<td><strong>xml2StartParse</strong></td>
<td>Element</td>
</tr>
</tbody>
</table>
**core**
Apache HTTP

**mpm_common**
(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**
fork

**mpm_winnt**
Windows NT

**worker**
mod_access_compat
   ( IP )
mod_actions
   CGI
mod_alias

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

mod_auth_digest
   User authentication using MD5 Digest Authentication
mod_auth_form
   Form authentication
mod_authn_anon
   "anonymous"
mod_authn_core
   Core Authentication
mod_authn_dbd
   User authentication using an SQL database
mod_authn_dbm
   DBM
mod_authn_file
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

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

mod_authz_owner

mod_authz_user

mod_autoindex
Unix ls Win32 dir

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

mod_buffer
Support for request buffering
mod_cache
URI

mod_cache_disk
URI

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

mod_cern_meta
CERN httpd metafile semantics

mod_cgi
CGI

mod_cgid
CGI CGI

mod_charset_lite
Specify character set translation or recoding

mod_data
Convert response body into an RFC2397 data URL

mod_dav
(WebDAV)

mod_dav_fs
mod_dav

mod_dav_lock
mod_dav

mod_dbd
Manages SQL database connections

mod_deflate

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

**mod_dir**

**mod_dumpio**
  I/O

**mod_echo**

**mod_env**
  CGI  SSI

**mod_example_hooks**
  Illustrates the Apache module API

**mod.expires**
  Expires  Cache-Control HTTP

**mod_ext_filter**

**mod_file_cache**
  Caches a static list of files in memory

**mod_filter**
  Context-sensitive smart filter configuration module

**mod_headers**
  HTTP

**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**
html (Server Side Includes)

**mod_info**

**mod_isapi**
ISAPI Extensions within Apache for Windows

**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**

**mod_log_debug**
Additional configurable debug logging

**mod_log_forensic**
forensic
mod_logio

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

modMacro
Provides macros within apache httpd runtime configuration files

mod_mime

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

mod_negotiation

mod_nw_ssl
Enable SSL encryption for NetWare

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

mod_proxy
HTTP/1.1 /

mod_proxy_ajp
mod_proxy AJP

mod_proxy_balancer
mod_proxy

mod_proxy_connect
CONNECT mod_proxy

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_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**

**mod_slotmem_plain**
Slot-based shared memory provider.

**mod_slotmem_shm**
Slot-based shared memory provider.

**mod_so**

**mod_socache_dbm**
DBM based shared object cache provider.
**mod sociache dc**
Distcache based shared object cache provider.

**mod sociache memcache**
Memcache based shared object cache provider.

**mod sociache shmcb**
shmcb based shared object cache provider.

**mod speling**
URL

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

**mod status**

**mod substitute**
Perform search and replace operations on response bodies

**mod suexec**
CGI

**mod unique id**

**mod unixd**
Basic (required) security for Unix-family platforms.

**mod userdir**

**mod usertrack**
*Clickstream* logging of user activity on a site

**mod version**

**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
Site Map

Apache HTTP 2.4
- 2.2
- Apache 2.3/2.4
- Apache 2.1/2.2
- Apache 2.0
- Apache License
- Apache
- Apache
- Directory, Location, Files
- Content Caching
- URL
- (DSO)
- Apache
- (MPM)
- Apache
- Apache
- suEXEC
- URL
- IP
- VirtualHost
- DNS_Apache
SSL/TLS FAQ
•

•

• **CGI**
• **Server Side Includes**
• **.htaccess**

•
- Microsoft Windows _Apache_
- Microsoft Windows _Apache_
- Novell NetWare _Apache_
- HPUX
- EBCDIC _Apache_
Apache HTTP:
- httpd
- ab
- apachectl
- apxs
- configure
- dbmmanage
- htcacheclean
- htdbm
- htdigest
- htpasswd
- logresolve
- rotatelogs
- suexec
- Apache
- Apache

- Apache
- Apache MPM
- Apache MPM event
- Apache MPM netware
- Apache MPM os2
- Apache MPM prefork
- Apache MPM winnt
- Apache MPM worker

- Apache_mod_access_compat
- Apache_mod_actions
- Apache_mod_alias
- Apache_mod_allowmethods
- Apache_mod_asis
- Apache_mod_auth_basic
- Apache_mod_auth_digest
- Apache_mod_auth_form
- Apache_mod_authn_anon
- Apache_mod_authn_core
- Apache_mod_authn_dbd
- Apache_mod_authn_dbm
- Apache_mod_authn_file
- Apache_mod_authn_socache
- Apache_mod_authnz_fcgid
- Apache_mod_authnz_idap
- Apache_mod_authz_core
- Apache_mod_authz_dbd
- Apache_mod_authz_dbm
- Apache_mod_authz_groupfile
- Apache_mod_authz_host
- Apache_mod_authz_owner
- Apache_mod_authz_user
- Apache_mod_autoindex
- Apache_mod_brotli
- Apache_mod_buffer
- Apache_mod_cache
- Apache_mod_cache_disk
- Apache_mod_cache_socache
- Apache_mod_cern_meta
- Apache_mod_cgi
- Apache_mod_cgid
- Apache_mod_charset_lite
- Apache_mod_data
- Apache_mod_dav
- Apache_mod_dav_fs
- Apache_mod_dav_lock
- Apache_mod_dbd
- Apache_mod_deflate
- Apache_mod_dialup
- Apache_mod_dir
- Apache_mod_dumpio
- Apache_mod_echo
- Apache_mod_env
- Apache_mod_example_hooks
- Apache_mod_expires
- Apache_mod_ext_filter
- Apache_mod_file_cache
- Apache_mod_filter
- Apache_mod_headers
- Apache_mod_heartbeat
- Apache_mod_heartmonitor
- Apache_mod_http2
- Apache_mod_ident
- Apache_mod_imagemap
• Apache_mod_include
• Apache_mod_info
• Apache_mod_isapi
• Apache_mod_lbmethod_bybusyness
• Apache_mod_lbmethod_byrequests
• Apache_mod_lbmethod_bytraffic
• Apache_mod_lbmethod_heartbeat
• Apache_mod_ldap
• Apache_mod_log_config
• Apache_mod_log_debug
• Apache_mod_log_forensic
• Apache_mod_logio
• Apache_mod_lua
• Apache_mod_macro
• Apache_mod_mime
• Apache_mod_mime_magic
• Apache_mod_negotiation
• Apache_mod_nw_ssl
• Apache_mod_privileges
• Apache_mod_proxy
• Apache_mod_proxy_aip
• Apache_mod_proxy_balancer
• Apache_mod_proxy_connect
• Apache_mod_proxy_express
• Apache_mod_proxy_fcgi
• Apache_mod_proxy_fdpass
• Apache_mod_proxy_ftp
• Apache_mod_proxy_hcheck
• Apache_mod_proxy_html
• Apache_mod_proxy_http
• Apache_mod_proxy_http2
• Apache_mod_proxy_scgi
• Apache_mod_proxy_wstunnel
• Apache_mod_ratelimit
- Apache_mod_reflector
- Apache_mod_remoteip
- Apache_mod_reqtimeout
- Apache_mod_request
- Apache_mod_rewrite
- Apache_mod_sed
- Apache_mod_session
- Apache_mod_session_cookie
- Apache_mod_session_crypto
- Apache_mod_session_dbd
- Apache_mod_setenvif
- Apache_mod_slotmem_plain
- Apache_mod_slotmem_shm
- Apache_mod_so
- Apache_mod_socache_dbm
- Apache_mod_socache_dc
- Apache_mod_socache_memcache
- Apache_mod_socache_shmcb
- Apache_mod_speling
- Apache_mod_ssl
- Apache_mod_status
- Apache_mod_substitute
- Apache_mod_suexec
- Apache_mod_unique_id
- Apache_mod_unixd
- Apache_mod_userdir
- Apache_mod_usertrack
- Apache_mod_version
- Apache_mod_vhost_alias
- Apache_mod_watchdog
- Apache_mod_xml2enc
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 `apExpr`, for all configuration directives. This document describes the `apExpr` expression parser.

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

See also

```
<If>
<ElseIf>
<Else>
ErrorDocument
Alias
ScriptAlias
Redirect
AuthBasicFake
AuthFormLoginRequiredLocation
AuthFormLoginSuccessLocation
AuthFormLogoutLocation
RewriteCond
SetEnvIfExpr
Header
```
RequestHeader
FilterProvider
Require expr
Require ldap-user
Require ldap-group
Require ldap-dn
Require ldap-attribute
Require ldap-filter
Require dbd-group
Require dbm-group
Require group
Require host
SSLRequire
LogMessage
mod include
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 `expr`. However, a few directives like `LogMessage` accept expressions that evaluate to a string value. For those, the starting point in the BNF is `string`.

```
expr ::= "true" | "false"
       | "!" expr
       | expr "&&" expr
       | expr "||" expr
       | "(" expr ")"
       | comp

comp ::= stringcomp
       | integercomp
       | unaryop word
       | word binaryop word
       | word "in" 
       | word "in" listfunction
       | word "=~" regex
       | word "!~" regex

comp ::= stringcomp
       | integercomp
       | unaryop word
       | word binaryop word
       | word "in" 
       | word "in" listfunction
       | word "=~" regex
       | word "!~" regex

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
```
wordlist ::= word
  | wordlist "," word

word ::= word "." word
  | digit
  | """" string """
  | """" string """
  | variable
  | rebackref
  | function

string ::= stringpart
  | string stringpart

stringpart ::= cstring
  | variable
  | rebackref

cstring ::= ...
digit ::= [0-9]+

variable ::= "%{" varname "}"
  | "%{" funcname ":" funcargs "}"

rebackref ::= "$" [0-9]

function ::= funcname "(" word ")"

listfunction ::= listfuncname "(" word ")"
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_ACCEPT</td>
<td></td>
</tr>
<tr>
<td>HTTP_COOKIE</td>
<td></td>
</tr>
<tr>
<td>HTTP_FORWARDED</td>
<td></td>
</tr>
<tr>
<td>HTTP_HOST</td>
<td></td>
</tr>
<tr>
<td>HTTP_PROXY_CONNECTION</td>
<td></td>
</tr>
<tr>
<td>HTTP_REFERER</td>
<td></td>
</tr>
<tr>
<td>HTTP_USER_AGENT</td>
<td></td>
</tr>
</tbody>
</table>

Other request related variables

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>REQUEST_METHOD</td>
<td>The HTTP method of the incoming request (e.g. GET)</td>
</tr>
<tr>
<td>REQUEST_SCHEME</td>
<td>The scheme part of the request's URI</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REQUEST_URI</td>
<td>The path part of the request's URI</td>
</tr>
<tr>
<td>DOCUMENT_URI</td>
<td>Same as REQUEST_URI</td>
</tr>
<tr>
<td>REQUEST_FILENAME</td>
<td>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</td>
</tr>
<tr>
<td>SCRIPT_FILENAME</td>
<td>Same as REQUEST_FILENAME</td>
</tr>
<tr>
<td>LAST_MODIFIED</td>
<td>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.</td>
</tr>
<tr>
<td>SCRIPT_USER</td>
<td>The user name of the owner of the script.</td>
</tr>
<tr>
<td>SCRIPT_GROUP</td>
<td>The group name of the group of the script.</td>
</tr>
<tr>
<td>PATH_INFO</td>
<td>The trailing path name information, see <code>AcceptPathInfo</code></td>
</tr>
<tr>
<td>QUERY_STRING</td>
<td>The query string of the current request</td>
</tr>
<tr>
<td>IS_SUBREQ</td>
<td>&quot;true&quot; if the current request is a subrequest, &quot;false&quot; otherwise</td>
</tr>
<tr>
<td>THE_REQUEST</td>
<td>The complete request line (e.g., &quot;GET /index.html HTTP/1.1&quot;)</td>
</tr>
<tr>
<td>REMOTE_ADDR</td>
<td>The IP address of the remote host</td>
</tr>
<tr>
<td>REMOTE_PORT</td>
<td>The port of the remote host (2.4.26)</td>
</tr>
<tr>
<td>Environment Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REMOTE_HOST</td>
<td>The host name of the remote host</td>
</tr>
<tr>
<td>REMOTE_USER</td>
<td>The name of the authenticated user, if any (not available during <code>&lt;If &gt;)</code></td>
</tr>
<tr>
<td>REMOTE_IDENT</td>
<td>The user name set by <code>mod_ident</code></td>
</tr>
<tr>
<td>SERVER_NAME</td>
<td>The <code>ServerName</code> of the current vhost</td>
</tr>
<tr>
<td>SERVER_PORT</td>
<td>The server port of the current vhost, see <code>ServerName</code></td>
</tr>
<tr>
<td>SERVER_ADMIN</td>
<td>The <code>ServerAdmin</code> of the current vhost</td>
</tr>
<tr>
<td>SERVER_PROTOCOL</td>
<td>The protocol used by the request</td>
</tr>
<tr>
<td>DOCUMENT_ROOT</td>
<td>The <code>DocumentRoot</code> of the current vhost</td>
</tr>
<tr>
<td>AUTH_TYPE</td>
<td>The configured <code>AuthType</code> (e.g. &quot;basic&quot;)</td>
</tr>
<tr>
<td>CONTENT_TYPE</td>
<td>The content type of the response (not available during <code>&lt;If &gt;</code>)</td>
</tr>
<tr>
<td>HANDLER</td>
<td>The name of the handler creating the response</td>
</tr>
<tr>
<td>HTTP2</td>
<td>&quot;on&quot; if the request uses http/2, &quot;off&quot; otherwise</td>
</tr>
<tr>
<td>HTTPS</td>
<td>&quot;on&quot; if the request uses https, &quot;off&quot; otherwise</td>
</tr>
<tr>
<td>IPV6</td>
<td>&quot;on&quot; if the connection uses IPv6, &quot;off&quot; otherwise</td>
</tr>
<tr>
<td>REQUEST_STATUS</td>
<td>The HTTP error status of the request (not available during <code>&lt;If &gt;</code>)</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>REQUEST_LOG_ID</td>
<td>The error log id of the request (see ErrorLogFormat)</td>
</tr>
<tr>
<td>CONN_LOG_ID</td>
<td>The error log id of the connection (see ErrorLogFormat)</td>
</tr>
<tr>
<td>CONN_REMOTE_ADDR</td>
<td>The peer IP address of the connection (see the mod_remoteip module)</td>
</tr>
<tr>
<td>CONTEXT_PREFIX</td>
<td></td>
</tr>
<tr>
<td>CONTEXT_DOCUMENT_ROOT</td>
<td></td>
</tr>
</tbody>
</table>

Misc variables

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Alternative</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>==</td>
<td>=</td>
<td>String equality</td>
</tr>
<tr>
<td>!=</td>
<td></td>
<td>String inequality</td>
</tr>
<tr>
<td>&lt;</td>
<td></td>
<td>String less than</td>
</tr>
<tr>
<td>&lt;=</td>
<td></td>
<td>String less than or equal</td>
</tr>
<tr>
<td>&gt;</td>
<td></td>
<td>String greater than</td>
</tr>
<tr>
<td>&gt;=</td>
<td></td>
<td>String greater than or equal</td>
</tr>
<tr>
<td>=~</td>
<td>!~</td>
<td>String matches the regular expression</td>
</tr>
<tr>
<td>!~</td>
<td></td>
<td>String does not match the regular expression</td>
</tr>
<tr>
<td>-eq</td>
<td>eq</td>
<td>Integer equality</td>
</tr>
<tr>
<td>-ne</td>
<td>ne</td>
<td>Integer inequality</td>
</tr>
<tr>
<td>-lt</td>
<td>lt</td>
<td>Integer less than</td>
</tr>
<tr>
<td>-le</td>
<td>le</td>
<td>Integer less than or equal</td>
</tr>
<tr>
<td>-gt</td>
<td>gt</td>
<td>Integer greater than</td>
</tr>
<tr>
<td>-ge</td>
<td>ge</td>
<td>Integer greater than or equal</td>
</tr>
</tbody>
</table>

**Other binary operators**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-ipmatch</td>
<td>IP address matches address/netmask</td>
</tr>
<tr>
<td>-strmatch</td>
<td>left string matches pattern given by right string (containing wildcards *, ?, [])</td>
</tr>
<tr>
<td>-</td>
<td>same as -strmatch, but case insensitive</td>
</tr>
<tr>
<td>strncmatch</td>
<td>-fnmatch</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Restricted</th>
</tr>
</thead>
<tbody>
<tr>
<td>-d</td>
<td>The argument is treated as a filename. True if the file exists and is a directory</td>
<td>yes</td>
</tr>
<tr>
<td>-e</td>
<td>The argument is treated as a filename. True if the file (or dir or special) exists</td>
<td>yes</td>
</tr>
<tr>
<td>-f</td>
<td>The argument is treated as a filename. True if the file exists and is regular file</td>
<td>yes</td>
</tr>
<tr>
<td>-s</td>
<td>The argument is treated as a filename. True if the file exists and is not empty</td>
<td>yes</td>
</tr>
<tr>
<td>-L</td>
<td>The argument is treated as a filename. True if the file exists and is symlink</td>
<td>yes</td>
</tr>
<tr>
<td>-h</td>
<td>The argument is treated as a filename. True if the file exists and is symlink (same as -L)</td>
<td>yes</td>
</tr>
<tr>
<td>-F</td>
<td>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!</td>
<td></td>
</tr>
<tr>
<td>-U</td>
<td>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!</td>
<td></td>
</tr>
<tr>
<td>-A</td>
<td>Alias for -U</td>
<td></td>
</tr>
<tr>
<td>-n</td>
<td>True if string is not empty</td>
<td></td>
</tr>
<tr>
<td>-z</td>
<td>True if string is empty</td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-------------------------</td>
<td></td>
</tr>
<tr>
<td>-T</td>
<td>False if string is empty, &quot;0&quot;, &quot;off&quot;, &quot;false&quot;, or &quot;no&quot; (case insensitive). True otherwise.</td>
<td></td>
</tr>
<tr>
<td>-R</td>
<td>Same as &quot;${REMOTE_ADDR} -ipmatch ...&quot;, but more efficient</td>
<td></td>
</tr>
</tbody>
</table>

The operators marked as "restricted" are not available in some modules like mod_include.
Normal string-valued functions take one string as argument and return a string. Functions names are not case sensitive. Modules may register additional functions.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Special notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>req, http</td>
<td>Get HTTP request header; header names may be added to the Vary header, see below</td>
<td></td>
</tr>
<tr>
<td>req_novary</td>
<td>Same as req, but header names will not be added to the Vary header</td>
<td></td>
</tr>
<tr>
<td>resp</td>
<td>Get HTTP response header</td>
<td></td>
</tr>
<tr>
<td>reqenv</td>
<td>Lookup request environment variable (as a shortcut, v can also be used to access variables).</td>
<td>ordering</td>
</tr>
<tr>
<td>osenv</td>
<td>Lookup operating system environment variable</td>
<td></td>
</tr>
<tr>
<td>note</td>
<td>Lookup request note</td>
<td>ordering</td>
</tr>
<tr>
<td>env</td>
<td>Return first match of note, reqenv, osenv</td>
<td>ordering</td>
</tr>
<tr>
<td>tolower</td>
<td>Convert string to lower case</td>
<td></td>
</tr>
<tr>
<td>toupper</td>
<td>Convert string to upper case</td>
<td></td>
</tr>
<tr>
<td>escape</td>
<td>Escape special characters in %hex encoding</td>
<td></td>
</tr>
<tr>
<td>unescape</td>
<td>Unescape %hex encoded string, leaving encoded slashes alone; return empty string if %00 is found</td>
<td></td>
</tr>
<tr>
<td>base64</td>
<td>Encode the string using base64 encoding</td>
<td></td>
</tr>
<tr>
<td>unbase64</td>
<td>Decode base64 encoded string, return truncated string if 0x00 is found</td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
<td>Availability</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
<td>--------------</td>
</tr>
<tr>
<td>md5</td>
<td>Hash the string using MD5, then encode the hash with hexadecimal encoding</td>
<td></td>
</tr>
<tr>
<td>sha1</td>
<td>Hash the string using SHA1, then encode the hash with hexadecimal encoding</td>
<td></td>
</tr>
<tr>
<td>file</td>
<td>Read contents from a file (including line endings, when present)</td>
<td>restricted</td>
</tr>
<tr>
<td>filemod</td>
<td>Return last modification time of a file (or 0 if file does not exist or is not regular file)</td>
<td>restricted</td>
</tr>
<tr>
<td>filesize</td>
<td>Return size of a file (or 0 if file does not exist or is not regular file)</td>
<td>restricted</td>
</tr>
</tbody>
</table>

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 \texttt{req} or \texttt{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 \texttt{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 wordlist, i.e. a list of strings. The wordlist can be used with the special \texttt{-in} operator. Functions names are not case sensitive. Modules may register additional functions.

There are no built-in list-valued functions. \texttt{mod\_ssl} provides \texttt{PeerExtList}. See the description of \texttt{SSLRequire} for details (but \texttt{PeerExtList} is also usable outside of \texttt{SSLRequire}).
The following examples show how expressions might be used to evaluate requests:

```html
# Compare the host name to example.com and redirect if it matches
<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 example in boolean context
<If "md5('foo') == 'acbd18db4cc2f85cedef654fccc4a4d8'">
  Header set checksum-matched true
</If>

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

# This delays the evaluation of the condition
Header always set CustomHeader my-value "expr=%{REQUEST_URI}~=/special_path\.php$"
<table>
<thead>
<tr>
<th>Name</th>
<th>Alternative</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-in</td>
<td>in</td>
<td>string contained in wordlist</td>
</tr>
<tr>
<td>/regexp/</td>
<td>m#regexp#</td>
<td>Regular expression (the second form allows different delimiters than /)</td>
</tr>
<tr>
<td>/regexp/i</td>
<td>m#regexp#i</td>
<td>Case insensitive regular expression</td>
</tr>
<tr>
<td>$0 ... $9</td>
<td></td>
<td>Regular expression backreferences</td>
</tr>
</tbody>
</table>

**Regular expression backreferences**

The strings $0 \ldots 9$ 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.
The `ap_expr` syntax is mostly a superset of the syntax of the deprecated `SSLRequire` directive. The differences are described in `SSLRequire`'s documentation.
The req_novary function is available for versions 2.4.4 and later.
Server and Supporting Programs

This page documents all the executable programs included with the Apache HTTP Server.
httpd
Apache hypertext transfer protocol server

apachectl
Apache HTTP server control interface

ab
Apache HTTP server benchmarking tool

apxs
APache eXtenSion tool

configure
Configure the source tree

dbmmanage
Create and update user authentication files in DBM format for basic authentication

fcgistarter
Start a FastCGI program

htcache
clean
Clean up the disk cache

htdigest
Create and update user authentication files for digest authentication

htdbm
Manipulate DBM password databases.

httpasswd
Create and update user authentication files for basic authentication

httxt2dbm
Create dbm files for use with RewriteMap

logresolve
Resolve hostnames for IP-addresses in Apache logfiles

**log server status**
Periodically log the server's status

**rotatelogs**
Rotate Apache logs without having to kill the server

**split-logfile**
Split a multi-vhost logfile into per-host logfiles

**suexec**
Switch User For Exec
Getting Started

If you're completely new to the Apache HTTP Server, or even to running a website at all, you might not know where to start, or what questions to ask. This document walks you through the basics.
Addresses on the Web are expressed with URLs - Uniform Resource Locators - which specify a protocol (e.g. http), a servername (e.g. www.apache.org), a URL-path (e.g. /docs/current/getting-started.html), and possibly a query string (e.g. ?arg=value) used to pass additional arguments to the server.

A client (e.g., a web browser) connects to a server (e.g., your Apache HTTP Server), with the specified protocol, and makes a request for a resource using the URL-path.

The URL-path may represent any number of things on the server. It may be a file (like getting-started.html) a handler (like server-status) or some kind of program file (like index.php). We'll discuss this more below in the Web Site Content section.

The server will send a response consisting of a status code and, optionally, a response body. The status code indicates whether the request was successful, and, if not, what kind of error condition there was. This tells the client what it should do with the response. You can read about the possible response codes in HTTP Server wiki.

Details of the transaction, and any error conditions, are written to log files. This is discussed in greater detail below in the Logs Files and Troubleshooting section.
In order to connect to a server, the client will first have to resolve the servername to an IP address - the location on the Internet where the server resides. Thus, in order for your web server to be reachable, it is necessary that the servername be in DNS.

If you don't know how to do this, you'll need to contact your network administrator, or Internet service provider, to perform this step for you.

More than one hostname may point to the same IP address, and more than one IP address can be attached to the same physical server. Thus, you can run more than one website on the same physical server, using a feature called virtual hosts.

If you are testing a server that is not Internet-accessible, you can put host names in your hosts file in order to do local resolution. For example, you might want to put a record in your hosts file to map a request for www.example.com to your local system, for testing purposes. This entry would look like:

```
127.0.0.1 www.example.com
```

A hosts file will probably be located at /etc/hosts or C:\Windows\system32\drivers\etc\hosts.

The Apache HTTP Server is configured via simple text files. These files may be located any of a variety of places, depending on how exactly you installed the server. Common locations for these files may be found in the httpd wiki. If you installed httpd from source, the default location of the configuration files is /usr/local/apache2/conf. The default configuration file is usually called httpd.conf. This, too, can vary in third-party distributions of the server.

The configuration is frequently broken into multiple smaller files, for ease of management. These files are loaded via the Include directive. The names or locations of these sub-files are not magical, and may vary greatly from one installation to another. Arrange and subdivide these files as makes the most sense to you. If the file arrangement you have by default doesn't make sense to you, feel free to rearrange it.

The server is configured by placing configuration directives in these configuration files. A directive is a keyword followed by one or more arguments that set its value.

The question of "Where should I put that directive?" is generally answered by considering where you want a directive to be effective. If it is a global setting, it should appear in the configuration file, outside of any <Directory>, <Location>, <VirtualHost>, or other section. If it is to apply only to a particular directory, then it should go inside a <Directory> section referring to that directory, and so on. See the Configuration Sections document for further discussion of these sections.

In addition to the main configuration files, certain directives may go in .htaccess files located in the content directories. .htaccess files are primarily for people who do not have access to the main server configuration file(s). You can read more about .htaccess
files in the `.htaccess howto`. 
Web site content can take many different forms, but may be broadly divided into static and dynamic content.

Static content is things like HTML files, image files, CSS files, and other files that reside in the filesystem. The DocumentRoot directive specifies where in your filesystem you should place these files. This directive is either set globally, or per virtual host. Look in your configuration file(s) to determine how this is set for your server.

Typically, a document called index.html will be served when a directory is requested without a file name being specified. For example, if DocumentRoot is set to /var/www/html and a request is made for http://www.example.com/work/, the file /var/www/html/work/index.html will be served to the client.

Dynamic content is anything that is generated at request time, and may change from one request to another. There are numerous ways that dynamic content may be generated. Various handlers are available to generate content. CGI programs may be written to generate content for your site.

Third-party modules like mod_php may be used to write code that does a variety of things. Many third-party applications, written using a variety of languages and tools, are available for download and installation on your Apache HTTP Server. Support of these third-party things is beyond the scope of this documentation, and you should find their documentation or other support forums to answer your questions about them.

→
As an Apache HTTP Server administrator, your most valuable assets are the log files, and, in particular, the error log. Troubleshooting any problem without the error log is like driving with your eyes closed.

The location of the error log is defined by the `ErrorLog` directive, which may be set globally, or per virtual host. Entries in the error log tell you what went wrong, and when. They often also tell you how to fix it. Each error log message contains an error code, which you can search for online for even more detailed descriptions of how to address the problem. You can also configure your error log to contain a log ID which you can then correlate to an access log entry, so that you can determine what request caused the error condition.

You can read more about logging in the [logs documentation](#).
Once you have the prerequisites under your belt, it's time to move on.

This document covers only the bare basics. We hope that this gets you started, but there are many other things that you might need to know.

- Download
- Install
- Configure
- Start
- Frequently Asked Questions
Apache SSL/TLS

Apache HTTP  mod_ssl  OpenSSL  Secure Sockets Layer
Transport Layer Security
mod_ssl Configuration How-To
Introduction To SSL
mod_ssl

Copyright 2017 The Apache Software Foundation.
Apache License, Version 2.0.
mod_rewrite provides a way to modify incoming URL requests, dynamically, based on regular expression rules. This allows you to map arbitrary URLs onto your internal URL structure in any way you like.

It supports an unlimited number of rules and an unlimited number of attached rule conditions for each rule to provide a really flexible and powerful URL manipulation mechanism. The URL manipulations can depend on various tests: server variables, environment variables, HTTP headers, time stamps, external database lookups, and various other external programs or handlers, can be used to achieve granular URL matching.

Rewrite rules can operate on the full URLs, including the path-info and query string portions, and may be used in per-server context (httpd.conf), per-virtualhost context (<VirtualHost> blocks), or per-directory context (.htaccess files and <Directory> blocks). The rewritten result can lead to further rules, internal sub-processing, external request redirection, or proxy passthrough, depending on what flags you attach to the rules.

Since mod_rewrite is so powerful, it can indeed be rather complex. This document supplements the reference documentation, and attempts to allay some of that complexity, and provide highly annotated examples of common scenarios that you may handle with mod_rewrite. But we also attempt to show you when you should not use mod_rewrite, and use other standard Apache features instead, thus avoiding this unnecessary complexity.

- mod_rewrite reference documentation
- Introduction to regular expressions and mod_rewrite
- Using mod_rewrite for redirection and remapping of URLs
- Using mod_rewrite to control access
- Dynamic virtual hosts with mod_rewrite
- Dynamic proxying with mod_rewrite
- Using RewriteMap
- Advanced techniques
- When NOT to use mod_rewrite
- RewriteRule Flags
- Technical details

See also
- mod_rewrite reference documentation
- Mapping URLs to the Filesystem
- mod_rewrite wiki
- Glossary

Copyright 2017 The Apache Software Foundation.
Licensed under the Apache License, Version 2.0.
Apache

1 (www.company1.com and www.company2.com)

Apache 1.1

Apache 1.3

mod_vhost_alias
• (IP)
• IP (IP)
• ( )
•
•
• `<VirtualHost>
  • NameVirtualHost
  • ServerName
  • ServerAlias
  • ServerPath

Apache

```
/usr/local/apache2/bin/httpd -S
```

Apache

<table>
<thead>
<tr>
<th>IP</th>
</tr>
</thead>
</table>
Using Apache With RPM Based Systems (Redhat / CentOS / Fedora)

While many distributions make Apache httpd available as operating system supported packages, it can sometimes be desirable to install and use the canonical version of Apache httpd on these systems, replacing the natively provided versions of the packages.

While the Apache httpd project does not currently create binary RPMs for the various distributions out there, it is easy to build your own binary RPMs from the canonical Apache httpd tarball.

This document explains how to build, install, configure and run Apache httpd 2.4 under Unix systems supporting the RPM packaging format.
The Apache httpd source tarball can be converted into an SRPM as follows:

```
rpmbuild -ts httpd-2.4.x.tar.bz2
```
RPMs can be built directly from the Apache httpd source tarballs using the following command:

```
rpmbuild -tb httpd-2.4.x.tar.bz2
```

Corresponding "-devel" packages will be required to be installed on your build system prior to building the RPMs, the rpmbuild command will automatically calculate what RPMs are required and will list any dependencies that are missing on your system. These "-devel" packages will not be required after the build is completed, and can be safely removed.

If successful, the following RPMs will be created:

- **httpd-2.4.x-1.i686.rpm**
  The core server and basic module set.

- **httpd-debuginfo-2.4.x-1.i686.rpm**
  Debugging symbols for the server and all modules.

- **httpd-devel-2.4.x-1.i686.rpm**
  Headers and development files for the server.

- **httpd-manual-2.4.x-1.i686.rpm**
  The webserver manual.

- **httpd-tools-2.4.x-1.i686.rpm**
  Supporting tools for the webserver.

- **mod_authnz_ldap-2.4.x-1.i686.rpm**
  `mod_ldap` and `mod_authnz_ldap`, with corresponding dependency on openldap.

- **mod_lua-2.4.x-1.i686.rpm**
  `mod_lua` module, with corresponding dependency on lua.

- **mod_proxy_html-2.4.x-1.i686.rpm**
  `mod_proxy_html` module, with corresponding dependency
on libxml2.

**mod_socache_dc-2.4.x-1.i686.rpm**

*mod_socache_dc* module, with corresponding dependency on distcache.

**mod_ssl-2.4.x-1.i686.rpm**

*mod_ssl* module, with corresponding dependency on openssl.
The `httpd` RPM is the only RPM necessary to get a basic server to run. Install it as follows:

```
rpm -U httpd-2.4.x-1.i686.rpm
```

Self contained modules are included with the server. Modules that depend on external libraries are provided as separate RPMs to install if needed.
The default configuration for the server is installed by default beneath the `/etc/httpd` directory, with logs written by default to `/var/log/httpd`. The environment for the webserver is set by default within the optional `/etc/sysconfig/httpd` file.

Start the server as follows:

```
service httpd restart
```
Configuring Additional Instances of Apache httpd on the Same Machine

It is possible to configure additional instances of the Apache httpd server running independently alongside each other on the same machine. These instances can have independent configurations, and can potentially run as separate users if so configured.

This was done by making the httpd startup script aware of its own name. This name is then used to find the environment file for the server, and in turn, the server root of the server instance.

To create an additional instance called httpd-additional, follow these steps:

- Create a symbolic link to the startup script for the additional server:

  ```
  ln -s /etc/rc.d/init.d/httpd /etc/rc.d/init.d/httpd-additional
  chkconfig --add httpd-additional
  ```

- Create an environment file for the server, using the `/etc/sysconfig/httpd` file as a template:

  ```
  # template from httpd
  cp /etc/sysconfig/httpd /etc/sysconfig/httpd-additional
  
  # blank template
  touch /etc/sysconfig/httpd-additional
  ```

  Edit `/etc/sysconfig/httpd-additional` and pass the server root of the new server instance within the OPTIONS environment variable.

  ```
  OPTIONS="-d /etc/httpd-additional -f conf/httpd-additional.conf"
  ```
- Edit the server configuration file `/etc/httpd-additional/conf/httpd-additional.conf` to ensure the correct ports and paths are configured.
- Start the server as follows:

  ```
  service httpd-additional restart
  ```

- Repeat this process as required for each server instance.
Warning
Many of the documents listed here are in need of update. They are in different stages of progress. Please be patient and follow this link to propose a fix or point out any error/discrepancy.
• Developing modules for the Apache HTTP Server 2.4
• Hook Functions in 2.4
• Request Processing in 2.4
• How filters work in 2.4
• Guidelines for output filters in 2.4
• Documenting code in 2.4
• Thread Safety Issues in 2.4
- API changes in 2.3/2.4
- Converting Modules from 1.3 to 2.x
- Autogenerated Apache HTTP Server (trunk) code documentation (the link is built by this job).
- Developer articles at [apachetutor](https://apachetutor) include:
  - Request Processing
  - Configuration for Modules
  - Resource Management
  - Connection Pooling
  - Introduction to Buckets and Brigades

Copyright 2017 The Apache Software Foundation. Licensed under the Apache License, Version 2.0.
Apache Miscellaneous Documentation

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

**Warning**

The documents below have not been fully updated to take into account changes made in the 2.1 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

**Performance Notes - Apache Tuning**

Notes about how to (run-time and compile-time) configure Apache for highest performance. Notes explaining why Apache does some things, and why it doesn't do other things (which make it slower/faster).

**Security Tips**

Some "do"s - and "don't"s - for keeping your Apache web site secure.

**Relevant Standards**

This document acts as a reference page for most of the relevant standards that Apache follows.

**Password Encryption Formats**

Discussion of the various ciphers supported by Apache for authentication purposes.
API Changes in Apache HTTP Server 2.4 since 2.2

This document describes changes to the Apache HTTPD API from version 2.2 to 2.4, that may be of interest to module/application developers and core hacks. As of the first GA release of the 2.4 branch API compatibility is preserved for the life of the 2.4 branch. (The VERSIONING description for the 2.4 release provides more information about API compatibility.)

API changes fall into two categories: APIs that are altogether new, and existing APIs that are expanded or changed. The latter are further divided into those where all changes are backwards-compatible (so existing modules can ignore them), and those that might require attention by maintainers. As with the transition from HTTPD 2.0 to 2.2, existing modules and applications will require recompiling and may call for some attention, but most should not require any substantial updating (although some may be able to take advantage of API changes to offer significant improvements).

For the purpose of this document, the API is split according to the public header files. These headers are themselves the reference documentation, and can be used to generate a browsable HTML reference with make docs.
ap_expr (NEW!)
Introduces a new API to parse and evaluate boolean and algebraic expressions, including provision for a standard syntax and customised variants.

ap_listen (changed; backwards-compatible)
Introduces a new API to enable httpd child processes to serve different purposes.

ap_mpm (changed)
ap_mpm_run is replaced by a new mpm hook. Also ap_graceful_stop_signalled is lost, and ap_mpm_register_timed_callback is new.

ap_regex (changed)
In addition to the existing regexp wrapper, a new higher-level API ap_rxplus is now provided. This provides the capability to compile Perl-style expressions like \texttt{s/regexp/replacement/flags} and to execute them against arbitrary strings. Support for regexp backreferences is also added.

ap_slotmem (NEW!)
Introduces an API for modules to allocate and manage memory slots, most commonly for shared memory.

ap_socache (NEW!)
API to manage a shared object cache.

heartbeat (NEW!)
common structures for heartbeat modules

ap_parse_htaccess (changed)
The function signature for ap_parse_htaccess has been changed. A apr_table_t of individual directives allowed for override must now be passed (override remains).

http_config (changed)
- Introduces per-module, per-directory loglevels, including macro wrappers.
- New AP_DECLARE_MODULE macro to declare all modules.
- New APLOG_USE_MODULE macro necessary for per-module loglevels in multi-file modules.
- New API to retain data across module unload/load
- New check_config hook
- New ap_process_fnmatch_configs() function to process wildcards
- Change ap_configfile_t, ap_cfg_getline(), ap_cfg_getc() to return error codes, and add ap_cfg_strerror() for retrieving an error description.
- Any config directive permitted in ACCESS_CONF context must now correctly handle being called from an .htaccess file via the new AllowOverrideList directive.
  ap_check_cmd_context() accepts a new flag NOT_IN_HTACCESS to detect this case.

http_core (changed)
- REMOVED ap_default_type, ap_requires, all 2.2 authnz API
- Introduces Optional Functions for logio and authnz
- New function ap_get_server_name_for_url to support IPv6 literals.
- New function `ap_register_errorlog_handler` to register error log format string handlers.
- Arguments of `error_log` hook have changed. Declaration has moved to `http_core.h`.
- New function `ap_state_query` to determine if the server is in the initial configuration preflight phase or not. This is both easier to use and more correct than the old method of creating a pool userdata entry in the process pool.
- New function `ap_get_conn_socket` to get the socket descriptor for a connection. This should be used instead of accessing the core connection config directly.

**httpd (changed)**

- Introduce per-directory, per-module loglevel
- New loglevels APLOG_TRACE
- Introduce errorlog ids for requests and connections
- Support for `mod_request kept_body`
- Support buffering filter data for async requests
- New `CONN_STATE` values
- Function changes: `ap_escape_html` updated; `ap_unescape_all`, `ap_escape_path_segment_buffer`
- Modules that load other modules later than the `EXEC_ON_READ` config reading stage need to call `ap_reserve_module_slots()` or `ap_reserve_module_slots_directive()` in their `pre_config` hook.
- The useragent IP address per request can now be tracked independently of the client IP address of the connection, for support of deployments with load balancers.

**http_log (changed)**

- Introduce per-directory, per-module loglevel
- New loglevels APLOG_TRACEn
- \texttt{ap\_log\_*error} become macro wrappers (backwards-compatible if APLOG_MARK macro is used, except that is no longer possible to use \texttt{#ifdef} inside the argument list)
- piped logging revamped
- \texttt{module\_index} added to error_log hook
- new function: \texttt{ap\_log\_command\_line}

\textbf{http\_request (changed)}

- New auth\_internal API and auth\_provider API
- New EOR bucket type
- New function \texttt{ap\_process\_async\_request}
- New flags \texttt{AP\_AUTH\_INTERNAL\_PER\_CONF} and \texttt{AP\_AUTH\_INTERNAL\_PER\_URI}
- New access\_checker\_ex hook to apply additional access control and/or bypass authentication.
- New functions \texttt{ap\_hook\_check\_access\_ex}, \texttt{ap\_hook\_check\_access}, \texttt{ap\_hook\_check\_authn}, \texttt{ap\_hook\_check\_authz} which accept \texttt{AP\_AUTH\_INTERNAL\_PER\_*} flags
- DEPRECATED direct use of \texttt{ap\_hook\_access\_checker}, \texttt{access\_checker\_ex}, \texttt{ap\_hook\_check\_user\_id}, \texttt{ap\_hook\_auth\_checker}

When possible, registering all access control hooks (including authentication and authorization hooks) using \texttt{AP\_AUTH\_INTERNAL\_PER\_CONF} is recommended. If all modules' access control hooks are registered with this flag, then whenever the server handles an internal sub-request that matches the same set of access control configuration directives as the initial request (which is the common case), it can avoid invoking the access control hooks another time.
If your module requires the old behavior and must perform access control checks on every sub-request with a different URI from the initial request, even if that URI matches the same set of access control configuration directives, then use AP_AUTH_INTERNAL_PER_URI.

**mod_auth (NEW!)**

Introduces the new provider framework for authn and authz

**mod_cache (changed)**

Introduces a commit_entity() function to the cache provider interface, allowing atomic writes to cache. Add a cache_status() hook to report the cache decision. All private structures and functions were removed.

**mod_core (NEW!)**

This introduces low-level APIs to send arbitrary headers, and exposes functions to handle HTTP OPTIONS and TRACE.

**mod_cache_disk (changed)**

Changes the disk format of the disk cache to support atomic cache updates without locking. The device/inode pair of the body file is embedded in the header file, allowing confirmation that the header and body belong to one another.

**mod_disk_cache (renamed)**

The mod_disk_cache module has been renamed to mod_cache_disk in order to be consistent with the naming of other modules within the server.

**mod_request (NEW!)**
The API for `mod_request`, to make input data available to multiple application/handler modules where required, and to parse HTML form data.

**mpm_common (changed)**

- REMOVES: accept, lockfile, lock_mech, set_scoreboard (locking uses the new `ap_mutex` API)
- NEW API to drop privileges (delegates this platform-dependent function to modules)
- NEW Hooks: `mpm_query`, `timed_callback`, and `get_name`
- CHANGED interfaces: `monitor` hook, `ap_reclaim_child_processes`, `ap_relieve_child_processes`

**scoreboard (changed)**

`ap_get_scoreboard_worker` is made non-backwards-compatible as an alternative version is introduced. Additional `proxy_balancer` support. Child status stuff revamped.

**util_cookies (NEW!)**

Introduces a new API for managing HTTP Cookies.

**util_ldap (changed)**

*no description available*

**util_mutex (NEW!)**

A wrapper for APR proc and global mutexes in `httpd`, providing common configuration for the underlying mechanism and location of lock files.
util_script (changed)
NEW: ap_args_to_table

util_time (changed)
NEW: ap_recent_ctime_ex
Logging

In order to take advantage of per-module log level configuration, any source file that calls the ap_log_* functions should declare which module it belongs to. If the module's module_struct is called foo_module, the following code can be used to remain backward compatible with HTTPD 2.0 and 2.2:

```c
#include <http_log.h>

#ifdef APLOG_USE_MODULE
APLOG_USE_MODULE(foo);
#endif
```

Note: This is absolutely required for C++-language modules. It can be skipped for C-language modules, though that breaks module-specific log level support for files without it.

The number of parameters of the ap_log_* functions and the definition of APLOG_MARK has changed. Normally, the change is completely transparent. However, changes are required if a module uses APLOG_MARK as a parameter to its own functions or if a module calls ap_log_* without passing APLOG_MARK. A module which uses wrappers around ap_log_* typically uses both of these constructs.

The easiest way to change code which passes APLOG_MARK to its own functions is to define and use a different macro that expands to the parameters required by those functions, as APLOG_MARK should only be used when calling ap_log_* directly. In this way, the code will remain compatible with HTTPD 2.0 and 2.2.

Code which calls ap_log_* without passing APLOG_MARK will necessarily differ between 2.4 and earlier releases, as 2.4 requires
a new third argument, APLOG_MODULE_INDEX.

/* code for httpd 2.0/2.2 */
ap_log_perror(file, line, APLOG_ERR, 0, p, "Failed to allocate
dynamic lock structure");

/* code for httpd 2.4 */
ap_log_perror(file, line, APLOG_MODULE_INDEX, APLOG_ERR, 0, p,
"Failed to allocate dynamic lock structure");

ap_log_*error are now implemented as macros. This means that it is no longer possible to use #ifdef inside the argument list of ap_log_*error, as this would cause undefined behavior according to C99.

A server_rec pointer must be passed to ap_log_error() when called after startup. This was always appropriate, but there are even more limitations with a NULL server_rec in 2.4 than in previous releases. Beginning with 2.3.12, the global variable ap_server_conf can always be used as the server_rec parameter, as it will be NULL only when it is valid to pass NULL to ap_log_error(). ap_server_conf should be used only when a more appropriate server_rec is not available.

Consider the following changes to take advantage of the new APLOG_TRACE1..8 log levels:

- Check current use of APLOG_DEBUG and consider if one of the APLOG_TRACEn levels is more appropriate.
- If your module currently has a mechanism for configuring the amount of debug logging which is performed, consider eliminating that mechanism and relying on the use of different APLOG_TRACEn levels. If expensive trace processing needs to be bypassed depending on the configured log level, use the APLOGtracen and APLOGrtracen macros to first check if
tracing is enabled.

Modules sometimes add process id and/or thread id to their log messages. These ids are now logged by default, so it may not be necessary for the module to log them explicitly. (Users may remove them from the error log format, but they can be instructed to add it back if necessary for problem diagnosis.)

If your module uses these existing APIs...

**ap_default_type()**
This is no longer available; Content-Type must be configured explicitly or added by the application.

**ap_get_server_name()**
If the returned server name is used in a URL, use ap_get_server_name_for_url() instead. This new function handles the odd case where the server name is an IPv6 literal address.

**ap_get_server_version()**
For logging purposes, where detailed information is appropriate, use ap_get_server_description(). When generating output, where the amount of information should be configurable by ServerTokens, use ap_get_server_banner().

**ap_graceful_stop_signalled()**
Replace with a call to ap_mpm_query(AP_MPMQ_MPM_STATE) and checking for state AP_MPMQ_STOPPING.

**ap_max_daemons_limit**, **ap_my_generation**, and **ap_threads_per_child**
Use ap_mpm_query() query codes AP_MPMQ_MAX_DAEMON_USED, AP_MPMQ_GENERATION, and AP_MPMQ_MAX_THREADS, respectively.
ap_mpm_query()
Ensure that it is not used until after the register-hooks hook has completed. Otherwise, an MPM built as a DSO would not have had a chance to enable support for this function.

ap_requires()
The core server now provides better infrastructure for handling Require configuration. Register an auth provider function for each supported entity using ap_register_auth_provider(). The function will be called as necessary during Require processing. (Consult bundled modules for detailed examples.)

ap_server_conf->process->pool userdata
Optional:
- If your module uses this to determine which pass of the startup hooks is being run, use ap_state_query(AP_SQ_MAIN_STATE).
- If your module uses this to maintain data across the unloading and reloading of your module, use ap_retained_data_create() and ap_retained_data_get().

apr_global_mutex_create(), apr_proc_mutex_create()
Optional: See ap_mutex_register(), ap_global_mutex_create(), and ap_proc_mutex_create(); these allow your mutexes to be configurable with the Mutex directive; you can also remove any configuration mechanisms in your module for such mutexes

CORE_PRIVATE
This is now unnecessary and ignored.

dav_new_error() and dav_new_error_tag()
Previously, these assumed that errno contained information
describing the failure. Now, an apr_status_t parameter must be provided. Pass 0/APR_SUCCESS if there is no such error information, or a valid apr_status_t value otherwise.

**mpm_default.h, DEFAULT_LOCKFILE, DEFAULT_THREAD_LIMIT, DEFAULT_PIDLOG, etc.**

The header file and most of the default configuration values set in it are no longer visible to modules. (Most can still be overridden at build time.) DEFAULT_PIDLOG and DEFAULT_REL_RUNTIMEDIR are now universally available via ap_config.h.

**unixd_config**

This has been renamed to ap_unixd_config.

**unixd_setup_child()**

This has been renamed to ap_unixd_setup_child(), but most callers should call the added ap_run_drop_privileges() hook.

**conn_rec->remote_ip and conn_rec->remote_addr**

These fields have been renamed in order to distinguish between the client IP address of the connection and the useragent IP address of the request (potentially overridden by a load balancer or proxy). References to either of these fields must be updated with one of the following options, as appropriate for the module:

- When you require the IP address of the user agent, which might be connected directly to the server, or might optionally be separated from the server by a transparent load balancer or proxy, use request_rec->useragent_ip and request_rec->useragent_addr.
- When you require the IP address of the client that is connected directly to the server, which might be the useragent or might be the load balancer or proxy itself, use conn_rec->client_ip and conn_rec-
If your module interfaces with this feature...

suEXEC
Optional: If your module logs an error when `ap_unixd_config.suexec_enabled` is 0, also log the value of the new field `suexec_disabled_reason`, which contains an explanation of why it is not available.

Extended status data in the scoreboard
In previous releases, ExtendedStatus had to be set to On, which in turn required that `mod_status` was loaded. In 2.4, just set `ap_extended_status` to 1 in a pre-config hook and the extended status data will be available.

Does your module...

Parse query args
Consider if `ap_args_to_table()` would be helpful.

Parse form data...
Use `ap_parse_form_data()`.

Check for request header fields Content-Length and Transfer-Encoding to see if a body was specified
Use `ap_request_has_body()`.

Implement cleanups which clear pointer variables
Use `ap_pool_cleanup_set_null()`.

Create run-time files such as shared memory files, pid files, etc.
Use `ap_runtime_dir_relative()` so that the global configuration for the location of such files, either by the DEFAULT_REL_RUNTIMEDIR compile setting or the `DefaultRuntimeDir` directive, will be respected. Apache
httpd 2.4.2 and above.

Copyright 2017 The Apache Software Foundation.
Licensed under the Apache License, Version 2.0.
fcgistarter - Start a FastCGI program

See also

mod_proxy_fcgi
Currently only works on Unix systems.
fcgistarter -c command -p port [ -i interface ] -N num
-c  command
    FastCGI program

-p  port
    Port which the program will listen on

-i  interface
    Interface which the program will listen on

-N  num
    Number of instances of the program
The Shared Object Cache provides a means to share simple data across all a server's workers, regardless of thread and process models. It is used where the advantages of sharing data across processes outweigh the performance overhead of inter-process communication.
The shared object cache as such is an abstraction. Four different modules implement it. To use the cache, one or more of these modules must be present, and configured.

The only configuration required is to select which cache provider to use. This is the responsibility of modules using the cache, and they enable selection using directives such as CacheSocache, AuthnCacheSOCache, SSLSessionCache, and SSLStaplingCache.

Currently available providers are:

"dbm" (mod_socache_dbm)
This makes use of a DBM hash file. The choice of underlying DBM used may be configurable if the installed APR version supports multiple DBM implementations.

"dc" (mod_socache_dc)
This makes use of the distcache distributed session caching libraries.

"memcache" (mod_socache_memcache)
This makes use of the memcached high-performance, distributed memory object caching system.

"shmcb" (mod_socache_shmcb)
This makes use of a high-performance cyclic buffer inside a shared memory segment.

The API provides the following functions:

const char *create(ap_socache_instance_t **instance, const char *arg, apr_pool_t *tmp, apr_pool_t *p);
Create a session cache based on the given configuration string. The instance pointer returned in the instance parameter will be passed as the first argument to subsequent
invocations.

`apr_status_t init(ap_socache_instance_t *instance, const char *cname, const struct ap_socache_hints *hints, server_rec *s, apr_pool_t *pool)`

Initialize the cache. The `cname` must be of maximum length 16 characters, and uniquely identifies the consumer of the cache within the server; using the module name is recommended, e.g. "mod_ssl-sess". This string may be used within a filesystem path so use of only alphanumeric [a-z0-9_-] characters is recommended. If `hints` is non-NULL, it gives a set of hints for the provider. Return APR error code.

`void destroy(ap_socache_instance_t *instance, server_rec *s)`

Destroy a given cache instance object.

`apr_status_t store(ap_socache_instance_t *instance, server_rec *s, const unsigned char *id, unsigned int idlen, apr_time_t expiry, unsigned char *data, unsigned int datalen, apr_pool_t *pool)`

Store an object in a cache instance.

`apr_status_t retrieve(ap_socache_instance_t *instance, server_rec *s, const unsigned char *id, unsigned int idlen, unsigned char *data, unsigned int *datalen, apr_pool_t *pool)`

Retrieve a cached object.

`apr_status_t remove(ap_socache_instance_t *instance, server_rec *s, const unsigned char *id, unsigned int idlen, apr_pool_t *pool)`

Remove an object from the cache.

`void status(ap_socache_instance_t *instance, request_rec *r, int flags)`

Dump the status of a cache instance for mod_status.

`apr_status_t iterate(ap_socache_instance_t *instance, server_rec *s, void *userctx, ap_socache_iterator_t *iterator,`
apr_pool_t *pool)
Dump all cached objects through an iterator callback.
httxt2dbm - Generate dbm files for use with RewriteMap

httxt2dbm is used to generate dbm files from text input, for use in RewriteMap with the dbm map type.

If the output file already exists, it will not be truncated. New keys will be added and existing keys will be updated.

See also

httpd
mod_rewrite
httxt2dbm [ -v ] [ -f DBM_TYPE ] -i SOURCE_TXT -o OUTPUT_DBM
-v
More verbose output

-f DBM_TYPE
Specify the DBM type to be used for the output. If not specified, will use the APR Default. Available types are: GDBM for GDBM files, SDBM for SDBM files, DB for berkeley DB files, NDBM for NDBM files, default for the default DBM type.

-i SOURCE_TXT
Input file from which the dbm is to be created. The file should be formatted with one record per line, of the form: key value. See the documentation for RewriteMap for further details of this file's format and meaning.

-o OUTPUT_DBM
Name of the output dbm files.
httxt2dbm -i rewritemap.txt -o rewritemap.dbm
httxt2dbm -f SDBM -i rewritemap.txt -o rewritemap.dbm
Platform Specific Notes
Using Apache
    This document explains how to install, configure and run Apache 2.4 under Microsoft Windows.

    See: Using Apache with Microsoft Windows

Compiling Apache
    There are many important points before you begin compiling Apache. This document explain them.

    See: Compiling Apache for Microsoft Windows
RPM Based Systems (Redhat / CentOS / Fedora)
This document explains how to build, install, and run Apache 2.4 on systems supporting the RPM packaging format.

See: Using Apache With RPM Based Systems

[Image]
Novell NetWare

This document explains how to install, configure and run Apache 2.4 under Novell NetWare 5.1 and above.

See: Using Apache With Novell NetWare

EBCDIC

Version 1.3 of the Apache HTTP Server is the first version which includes a port to a (non-ASCII) mainframe machine which uses the EBCDIC character set as its native codeset.

**Warning:** This document has not been updated to take into account changes made in the 2.4 version of the Apache HTTP Server. Some of the information may still be relevant, but please use it with care.

See: The Apache EBCDIC Port

Copyright 2017 The Apache Software Foundation.
Licensed under the Apache License, Version 2.0.
suexec is used by the Apache HTTP Server to switch to another user before executing CGI programs. In order to achieve this, it must run as root. Since the HTTP daemon normally doesn't run as root, the suexec executable needs the setuid bit set and must be owned by root. It should never be writable for any other person than root.

For further information about the concepts and the security model of suexec please refer to the suexec documentation (http://httpd.apache.org/docs/2.4/suexec.html).
suexec -V
-V

If you are root, this option displays the compile options of suexec. For security reasons all configuration options are changeable only at compile time.
Dynamic mass virtual hosts with mod_rewrite

This document supplements the mod_rewrite reference documentation. It describes how you can use mod_rewrite to create dynamically configured virtual hosts.

mod_rewrite is not the best way to configure virtual hosts. You should first consider the alternatives before resorting to mod_rewrite. See also the "how to avoid mod_rewrite" document.

See also

Module documentation
mod_rewrite introduction
Redirection and remapping
Controlling access
Proxying
RewriteMap
Advanced techniques
When not to use mod_rewrite
Description:
We want to automatically create a virtual host for every hostname which resolves in our domain, without having to create new VirtualHost sections.

In this recipe, we assume that we'll be using the hostname www.SITE.example.com for each user, and serve their content out of /home/SITE/www.

Solution:

```plaintext
RewriteEngine on
RewriteMap lowercase int:tolower
RewriteCond %{lowercase:%{HTTP_HOST}} ^www.
RewriteRule ^(.*)$ /home/%1/www$1
```

Discussion

You will need to take care of the DNS resolution - Apache does not handle name resolution. You'll need either to create CNAME records for each hostname, or a DNS wildcard record. Creating DNS records is beyond the scope of this document.

The internal tolower RewriteMap directive is used to ensure that the hostnames being used are all lowercase, so that there is no ambiguity in the directory structure which must be created.

Parentheses used in a RewriteCond are captured into the
backreferences %1, %2, etc, while parentheses used in
RewriteRule are captured into the backreferences $1, $2, etc.

As with many techniques discussed in this document, mod_rewrite really isn't the best way to accomplish this task. You should, instead, consider using mod_vhost_alias instead, as it will much more gracefully handle anything beyond serving static files, such as any dynamic content, and Alias resolution.
This extract from httpd.conf does the same thing as the first example. The first half is very similar to the corresponding part above, except for some changes, required for backward compatibility and to make the mod_rewrite part work properly; the second half configures mod_rewrite to do the actual work.

Because mod_rewrite runs before other URI translation modules (e.g., mod_alias), mod_rewrite must be told to explicitly ignore any URLs that would have been handled by those modules. And, because these rules would otherwise bypass any ScriptAlias directives, we must have mod_rewrite explicitly enact those mappings.

```
# get the server name from the Host: header
UseCanonicalName Off

# splittable logs
LogFormat "%{Host}i %h %l %u %t "%r" %s %b" vcommon
CustomLog "logs/access_log" vcommon

<Directory "/www/hosts">
    # ExecCGI is needed here because we can't
    # CGI execution in the way that ScriptAlias
d    Options FollowSymLinks ExecCGI
</Directory>

RewriteEngine On

# a ServerName derived from a Host: header
RewriteMap lowercase int:tolower

### deal with normal documents first:
# allow Alias "/icons/" to work - repeat for
RewriteCond "%{REQUEST_URI}" "!^/icons/
# allow CGIs to work
```
RewriteCond "%{REQUEST_URI}" "!/cgi-bin/"
# do the magic
RewriteRule "^/(.*)$" "/www/hosts/${lowercase:%{SERVER_NAME}}/docs/$1"

## and now deal with CGIs - we have to force
RewriteCond "%{REQUEST_URI}" "^/cgi-bin/"
RewriteRule "^/(.*)$" "/www/hosts/${lowercase:%{SERVER_NAME}}/cgi-bin/$1"
This arrangement uses more advanced mod rewrite features to work out the translation from virtual host to document root, from a separate configuration file. This provides more flexibility, but requires more complicated configuration.

The vhost.map file should look something like this:

```
customer-1.example.com /www/customers/1
customer-2.example.com /www/customers/2
# ...
customer-N.example.com /www/customers/N
```

The httpd.conf should contain the following:

```
RewriteEngine on

RewriteMap lowercase int:tolower

# define the map file
RewriteMap vhost "txt:/www/conf/vhost.map"

# deal with aliases as above
RewriteCond %{REQUEST_URI} "^/icons/"
RewriteCond %{REQUEST_URI} "^/cgi-bin/"
RewriteCond ${lowercase:%{SERVER_NAME}} "^(.+)$"
# this does the file-based remap
RewriteCond %{vhost:%1} "^(/.*)$"
RewriteRule ^/(.*)$ "%/docs/$1"

RewriteCond %{REQUEST_URI} "^/cgi-bin/(.*)$"
RewriteCond %{lowercase:%{SERVER_NAME}} "^(.+)$"
RewriteCond %{vhost:%1} "^(/.*)$"
RewriteRule ^/cgi-bin/(.*)$ "%/cgi-bin/$1" [H=cgi-script]
```
CGI

CGI (Common Gateway Interface)

Apache

.htaccess

.htaccess

Server Side Includes

SSI (Server Side Includes) HTML

HTML

UserDir

http://example.com/~username/

UserDir

( public_html)
Access control refers to any means of controlling access to any resource. This is separate from authentication and authorization.
Access control can be done by several different modules. The most important of these are `mod_authz_core` and `mod_authz_host`. Also discussed in this document is access control using `mod_rewrite`. 
If you wish to restrict access to portions of your site based on the host address of your visitors, this is most easily done using mod_authz_host.

The Require provides a variety of different ways to allow or deny access to resources. In conjunction with the RequireAll, RequireAny, and RequireNone directives, these requirements may be combined in arbitrarily complex ways, to enforce whatever your access policy happens to be.

The Allow, Deny, and Order directives, provided by mod_access_compat, are deprecated and will go away in a future version. You should avoid using them, and avoid outdated tutorials recommending their use.

The usage of these directives is:

```plaintext
Require host address
Require ip ip.address
```

In the first form, address is a fully qualified domain name (or a partial domain name); you may provide multiple addresses or domain names, if desired.

In the second form, ip.address is an IP address, a partial IP address, a network/netmask pair, or a network/nnn CIDR specification. Either IPv4 or IPv6 addresses may be used.

See the mod_authz_host documentation for further examples of this syntax.

You can insert not to negate a particular requirement. Note, that since a not is a negation of a value, it cannot be used by itself to
allow or deny a request, as *not true* does not constitute *false*. Thus, to deny a visit using a negation, the block must have one element that evaluates as true or false. 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 (10.252.46.165) 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.

```
Require not host host.example.com
```

And, if you'd like to block access from an entire domain, you can specify just part of an address or domain name:

```
Require not ip 192.168.205
Require not host phishers.example.com moreidiots.example
Require not host gov
```

Use of the **RequireAll**, **RequireAny**, and **RequireNone** directives may be used to enforce more complex sets of requirements.
Using the `<If>`, you can allow or deny access based on arbitrary environment variables or request header values. For example, to deny access based on user-agent (the browser type) you might do the following:

```
<If "${HTTP_USER_AGENT} == 'BadBot'">
  Require all denied
</If>
```

Using the `Require expr` syntax, this could also be written as:

```
Require expr ${HTTP_USER_AGENT} != 'BadBot'
```

**Warning:**

Access control by User-Agent is an unreliable technique, since the User-Agent header can be set to anything at all, at the whim of the end user.

See [the expressions document](#) for a further discussion of what expression syntaxes and variables are available to you.
The [F] RewriteRule flag causes a 403 Forbidden response to be sent. Using this, you can deny access to a resource based on arbitrary criteria.

For example, if you wish to block access to a resource between 8pm and 7am, you can do this using mod_rewrite.

```
RewriteEngine On
RewriteCond %{TIME_HOUR} ">=20" [OR]
RewriteCond %{TIME_HOUR} "<07"
RewriteRule ^/fridge "-" [F]
```

This will return a 403 Forbidden response for any request after 8pm or before 7am. This technique can be used for any criteria that you wish to check. You can also redirect, or otherwise rewrite these requests, if that approach is preferred.

The <If> directive, added in 2.4, replaces many things that mod_rewrite has traditionally been used to do, and you should probably look there first before resorting to mod_rewrite.
The expression engine gives you a great deal of power to do a variety of things based on arbitrary server variables, and you should consult that document for more detail.

Also, you should read the mod_authz_core documentation for examples of combining multiple access requirements and specifying how they interact.

See also the Authentication and Authorization howto.
log_server_status - Log periodic status summaries

This perl script is designed to be run at a frequent interval by something like cron. It connects to the server and downloads the status information. It reformats the information to a single line and logs it to a file. Adjust the variables at the top of the script to specify the location of the resulting logfile. mod_status will need to be loaded and configured in order for this script to do its job.
The script contains the following section.

```
my $wherelog = "/usr/local/apache2/logs/";
my $server = "localhost";  # Name of server, could be "www.foo.com"
my $port = "80";           # Port on server
my $request = "/server-status/?auto";  # Request to send
```

You'll need to ensure that these variables have the correct values, and you'll need to have the `server-status` handler configured at the location specified, and the specified log location needs to be writable by the user which will run the script.

Run the script periodically via cron to produce a daily log file, which can then be used for statistical analysis.
This perl script will take a combined Web server access log file and break its contents into separate files. It assumes that the first field of each line is the virtual host identity, put there using the "%v" variable in LogFormat.
Create a log file with virtual host information in it:

```bash
LogFormat "%v %h %l %u %t "%r" %>s %>b "%{Referer}i" "%{User-agent}i"
CustomLog logs/access_log combined_plus_vhost

split-logfile < access_log
```

Log files will be created, in the directory where you run the script, for each virtual host name that appears in the combined log file. These logfiles will named after the hostname, with a .log file extension.

The combined log file is read from stdin. Records read will be appended to any existing log files.
Password Formats

Notes about the password encryption formats generated and understood by Apache.
There are five formats that Apache recognizes for basic-authentication passwords. Note that not all formats work on every platform:

**bcrypt**
"$2y$" + the result of the crypt_blowfish algorithm. See the APR source file `crypt_blowfish.c` for the details of the algorithm.

**MD5**
"$apr1$" + the result of an Apache-specific algorithm using an iterated (1,000 times) MD5 digest of various combinations of a random 32-bit salt and the password. See the APR source file `apr_md5.c` for the details of the algorithm.

**SHA1**
"{SHA}" + Base64-encoded SHA-1 digest of the password. Insecure.

**CRYPT**
Unix only. Uses the traditional Unix `crypt(3)` function with a randomly-generated 32-bit salt (only 12 bits used) and the first 8 characters of the password. Insecure.

**PLAIN TEXT (i.e. unencrypted)**
Windows & Netware only. Insecure.

### Generating values with htpasswd

**bcrypt**
```
$ htpasswd -nbB myName myPassword
myName:$2y$05$c4WoMPo3SXsafkva.HHa6uXQZWr7oboPiC2bT/r7q1BB8I2s0BF
```

**MD5**
```
$ htpasswd -nbm myName myPassword
myName:$apr1$r31.....$HqJZimcKQFAMYayBlzkrA/
```
SHA1

$ htpasswd -nbs myName myPassword
myName:{SHA}VBPuJHI7uixaa6LQGwx4s+5GKNE=

CRYPT

$ htpasswd -nbd myName myPassword
myName:rqXexS6ZhobKA

Generating CRYPT and MD5 values with the OpenSSL command-line program

OpenSSL knows the Apache-specific MD5 algorithm.

MD5

$ openssl passwd -apr1 myPassword
$apr1$qHDFfhPC$nITSVHgYbDAK1Y0acGRnY0

CRYPT

openssl passwd -crypt myPassword
qQ5vTY03c8dsU

Validating CRYPT or MD5 passwords with the OpenSSL command line program

The salt for a CRYPT password is the first two characters (converted to a binary value). To validate myPassword against rqXexS6ZhobKA

CRYPT

$ openssl passwd -crypt -salt rq myPassword
Warning: truncating password to 8 characters
rqXexS6ZhobKA
Note that using myPasswo instead of myPassword will produce the same result because only the first 8 characters of CRYPT passwords are considered.

The salt for an MD5 password is between $apr1$ and the following $ (as a Base64-encoded binary value - max 8 chars). To validate myPassword against $apr1$r31.....$HqJZimcKQFAMYayBlzkrA/

**MD5**

$ openssl passwd -apr1 -salt r31..... myPassword
$apr1$r31.....$HqJZimcKQFAMYayBlzkrA/

**Database password fields for mod_dbd**

The SHA1 variant is probably the most useful format for DBD authentication. Since the SHA1 and Base64 functions are commonly available, other software can populate a database with encrypted passwords that are usable by Apache basic authentication.

To create Apache SHA1-variant basic-authentication passwords in various languages:

**PHP**

'\{SHA\}' . base64_encode(sha1($password, TRUE))

**Java**

"{SHA}" + new
sun.misc.BASE64Encoder().encode(java.security.MessageDigest.getInstance("SHA1").digest(password.getBytes()))

**ColdFusion**

"{SHA}" & ToBase64(BinaryDecode(Hash(password, "SHA1"), "Hex"))
Ruby

```
require 'digest/sha1'
require 'base64'
'{SHA}' + Base64.encode64(Digest::SHA1.digest(password))
```

C or C++

Use the APR function: apr_sha1_base64

Python

```
import base64
import hashlib
"{SHA}" +
format(base64.b64encode(hashlib.sha1(password).digest()))
```

PostgreSQL (with the contrib/pgcrypto functions installed)

```
'{SHA}'||encode(digest(password, 'sha1'), 'base64')
```
Apache recognizes one format for digest-authentication passwords - the MD5 hash of the string user:realm:password as a 32-character string of hexadecimal digits. realm is the Authorization Realm argument to the AuthName directive in httpd.conf.

**Database password fields for mod_dbd**

Since the MD5 function is commonly available, other software can populate a database with encrypted passwords that are usable by Apache digest authentication.

To create Apache digest-authentication passwords in various languages:

**PHP**

```php
md5($user . ':' . $realm . ':' . $password)
```

**Java**

```java
byte b[] =
    java.security.MessageDigest.getInstance("MD5").digest( (user + ":" + realm + ":" + password ).getBytes());
java.math.BigInteger bi = new java.math.BigInteger(1, b);
String s = bi.toString(16);
while (s.length() < 32)
    s = "0" + s;
// String s is the encrypted password
```

**ColdFusion**

```coldfusion
LCase(Hash( (user & ":" & realm & ":" & password), "MD5")
```

**Ruby**

```ruby
require 'digest/md5'
Digest::MD5.hexdigest(user + ':' + realm + ':' + password)
```
PostgreSQL (with the contrib/pgcrypto functions installed)

```
encode(digest(user || ':' || realm || ':' || password, 'md5'), 'hex')
```
Using RewriteMap

This document supplements the [mod_rewrite reference documentation](https://httpd.apache.org/docs/). It describes the use of the [RewriteMap](https://httpd.apache.org/docs/) directive, and provides examples of each of the various RewriteMap types.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

See also

- [Module documentation](https://httpd.apache.org/docs/)
- [mod_rewrite introduction](https://httpd.apache.org/docs/)
- [Redirection and remapping](https://httpd.apache.org/docs/)
- [Controlling access](https://httpd.apache.org/docs/)
- [Virtual hosts](https://httpd.apache.org/docs/)
- [Proxying](https://httpd.apache.org/docs/)
- [Advanced techniques](https://httpd.apache.org/docs/)
- [When not to use mod_rewrite](https://httpd.apache.org/docs/)
The **RewriteMap** directive defines an external function which can be called in the context of **RewriteRule** or **RewriteCond** directives to perform rewriting that is too complicated, or too specialized to be performed just by regular expressions. The source of this lookup can be any of the types listed in the sections below, and enumerated in the **RewriteMap** reference documentation.

The syntax of the **RewriteMap** directive is as follows:

```
RewriteMap  MapName  MapType:MapSource
```

The *MapName* is an arbitrary name that you assign to the map, and which you will use in directives later on. Arguments are passed to the map via the following syntax:

```
${ MapName : LookupKey } ${ MapName : LookupKey | DefaultValue }
```

When such a construct occurs, the map *MapName* is consulted and the key *LookupKey* is looked-up. If the key is found, the map-function construct is substituted by *SubstValue*. If the key is not found then it is substituted by *DefaultValue* or by the empty string if no *DefaultValue* was specified.

For example, you can define a **RewriteMap** as:

```
RewriteMap examplemap "txt:/path/to/file/map.txt"
```

You would then be able to use this map in a **RewriteRule** as follows:
A default value can be specified in the event that nothing is found in the map:

```
RewriteRule "^/ex/(.*)" "${examplemap:$1|/not_found.html}"
```

**Per-directory and .htaccess context**

The `RewriteMap` directive may not be used in `<Directory>` sections or `.htaccess` files. You must declare the map in server or virtualhost context. You may use the map, once created, in your `RewriteRule` and `RewriteCond` directives in those scopes. You just can't `declare` it in those scopes.

The sections that follow describe the various `MapTypes` that may be used, and give examples of each.
When a MapType of `int` is used, the MapSource is one of the available internal `RewriteMap` functions. Module authors can provide additional internal functions by registering them with the `ap_register_rewrite_mapfunc` API. The functions that are provided by default are:

- **toupper**: Converts the key to all upper case.
- **tolower**: Converts the key to all lower case.
- **escape**: Translates special characters in the key to hex-encodings.
- **unescape**: Translates hex-encodings in the key back to special characters.

To use one of these functions, create a `RewriteMap` referencing the `int` function, and then use that in your `RewriteRule`:

**Redirect a URI to an all-lowercase version of itself**

```
RewriteMap lc int:tolower
RewriteRule "(.*)" "${lc:$1}" [R]
```

Please note that the example offered here is for illustration purposes only, and is not a recommendation. If you want to make URLs case-insensitive, consider using `mod_speling` instead.
When a MapType of `txt` is used, the MapSource is a filesystem path to a plain-text mapping file, containing one space-separated key/value pair per line. Optionally, a line may contain a comment, starting with a `#` character.

A valid text rewrite map file will have the following syntax:

```
# Comment line
MatchingKey SubstValue
MatchingKey SubstValue # comment
```

When the `RewriteMap` is invoked the argument is looked for in the first argument of a line, and, if found, the substitution value is returned.

For example, we can use a mapfile to translate product names to product IDs for easier-to-remember URLs, using the following recipe:

**Product to ID configuration**

```
RewriteMap product2id "txt:/etc/apache2/productmap.txt"
RewriteRule "^/product/(.*)" " /prods.php?id=${product2id:$1|NOTFOUND}" [PT]
```

We assume here that the `prods.php` script knows what to do when it received an argument of `id=NOTFOUND` when a product is not found in the lookup map.

The file `/etc/apache2/productmap.txt` then contains the following:

**Product to ID map**

```
##
## productmap.txt - Product to ID map file
```
Thus, when http://example.com/product/television is requested, the RewriteRule is applied, and the request is internally mapped to /prods.php?id=993.

Note: .htaccess files
The example given is crafted to be used in server or virtualhost scope. If you're planning to use this in a .htaccess file, you'll need to remove the leading slash from the rewrite pattern in order for it to match anything:

```apache
RewriteRule "^product/(.*)" /prods.php?id=${product2id:$1|NOTFOUND} [PT]
```

Cached lookups
The looked-up keys are cached by httpd until the mtime (modified time) of the mapfile changes, or the httpd server is restarted. This ensures better performance on maps that are called by many requests.
When a MapType of `rnd` is used, the MapSource is a filesystem path to a plain-text mapping file, each line of which contains a key, and one or more values separated by `|`. One of these values will be chosen at random if the key is matched.

For example, you can use the following map file and directives to provide a random load balancing between several back-end servers, via a reverse-proxy. Images are sent to one of the servers in the 'static' pool, while everything else is sent to one of the 'dynamic' pool.

### Rewrite map file

```plaintext
##
## map.txt -- rewriting map
##

static www1|www2|www3|www4
dynamic www5|www6
```

### Configuration directives

```plaintext
RewriteMap servers "rnd:/path/to/file/map.txt"

RewriteRule "^/(.*\.(png|gif|jpg))" "http://$\{servers:static\}/$1"
RewriteRule "^/(.*)" "http://$\{servers:dynamic\}/$1"
```

So, when an image is requested and the first of these rules is matched, `RewriteMap` looks up the string `static` in the map file, which returns one of the specified hostnames at random, which is then used in the `RewriteRule` target.

If you wanted to have one of the servers more likely to be chosen (for example, if one of the server has more memory than the others, and so can handle more requests) simply list it more times
in the map file.

```
static www1|www1|www2|www3|www4
```
When a MapType of dbm is used, the MapSource is a filesystem path to a DBM database file containing key/value pairs to be used in the mapping. This works exactly the same way as the txt map, but is much faster, because a DBM is indexed, whereas a text file is not. This allows more rapid access to the desired key.

You may optionally specify a particular dbm type:

```
RewriteMap examplemap "dbm=sdbm:/etc/apache/mapfile.dbm"
```

The type can be sdbm, gdbm, ndbm or db. However, it is recommended that you just use the `httxt2dbm` utility that is provided with Apache HTTP Server, as it will use the correct DBM library, matching the one that was used when httpd itself was built.

To create a dbm file, first create a text map file as described in the `txt` section. Then run `httxt2dbm`:

```
$ httxt2dbm -i mapfile.txt -o mapfile.map
```

You can then reference the resulting file in your `RewriteMap` directive:

```
RewriteMap mapname "dbm:/etc/apache/mapfile.map"
```

Note that with some dbm types, more than one file is generated, with a common base name. For example, you may have two files named `mapfile.map.dir` and `mapfile.map.pag`. This is normal, and you need only use the base name `mapfile.map` in your `RewriteMap` directive.
Cached lookups

The looked-up keys are cached by httpd until the mtime (modified time) of the mapfile changes, or the httpd server is restarted. This ensures better performance on maps that are called by many requests.
When a MapType of prg is used, the MapSource is a filesystem path to an executable program which will providing the mapping behavior. This can be a compiled binary file, or a program in an interpreted language such as Perl or Python.

This program is started once, when the Apache HTTP Server is started, and then communicates with the rewriting engine via STDIN and STDOUT. That is, for each map function lookup, it expects one argument via STDIN, and should return one new-line terminated response string on STDOUT. If there is no corresponding lookup value, the map program should return the four-character string "NULL" to indicate this.

External rewriting programs are not started if they're defined in a context that does not have RewriteEngine set to on.

This feature utilizes the rewrite-map mutex, which is required for reliable communication with the program. The mutex mechanism and lock file can be configured with the Mutex directive.

A simple example is shown here which will replace all dashes with underscores in a request URI.

**Rewrite configuration**

```conf
RewriteMap d2u "prg:/www/bin/dash2under.pl"
RewriteRule "-" "$d2u:%{REQUEST_URI}"
```

**dash2under.pl**

```perl
#!/usr/bin/perl
$|=1; # Turn off I/O buffering
```
Caution!

- Keep your rewrite map program as simple as possible. If the program hangs, it will cause httpd to wait indefinitely for a response from the map, which will, in turn, cause httpd to stop responding to requests.
- Be sure to turn off buffering in your program. In Perl this is done by the second line in the example script: $| = 1; This will of course vary in other languages. Buffered I/O will cause httpd to wait for the output, and so it will hang.
- Remember that there is only one copy of the program, started at server startup. All requests will need to go through this one bottleneck. This can cause significant slowdowns if many requests must go through this process, or if the script itself is very slow.
When a MapType of dbd or fast dbd is used, the MapSource is a SQL SELECT statement that takes a single argument and returns a single value.

`mod_dbd` will need to be configured to point at the right database for this statement to be executed.

There are two forms of this MapType. Using a MapType of `dbd` causes the query to be executed with each map request, while using `fast dbd` caches the database lookups internally. So, while `fast dbd` is more efficient, and therefore faster, it won’t pick up on changes to the database until the server is restarted.

If a query returns more than one row, a random row from the result set is used.

**Example**

```
RewriteMap myquery "fast dbd:SELECT destination FROM rewrite WHERE source = %s"
```
The `RewriteMap` directive can occur more than once. For each mapping-function use one `RewriteMap` directive to declare its rewriting mapfile.

While you cannot **declare** a map in per-directory context (`.htaccess` files or `<Directory>` blocks) it is possible to **use** this map in per-directory context.
Apache mod_rewrite Introduction

This document supplements the mod_rewrite reference documentation. It describes the basic concepts necessary for use of mod_rewrite. Other documents go into greater detail, but this doc should help the beginner get their feet wet.

See also

Module documentation
Redirection and remapping
Controlling access
Virtual hosts
Proxying
Using RewriteMap
Advanced techniques
When not to use mod_rewrite
The Apache module `mod_rewrite` is a very powerful and sophisticated module which provides a way to do URL manipulations. With it, you can do nearly all types of URL rewriting that you may need. It is, however, somewhat complex, and may be intimidating to the beginner. There is also a tendency to treat rewrite rules as magic incantation, using them without actually understanding what they do.

This document attempts to give sufficient background so that what follows is understood, rather than just copied blindly.

Remember that many common URL-manipulation tasks don't require the full power and complexity of `mod_rewrite`. For simple tasks, see `mod_alias` and the documentation on mapping URLs to the filesystem.

Finally, before proceeding, be sure to configure `mod_rewrite`'s log level to one of the trace levels using the `LogLevel` directive. Although this can give an overwhelming amount of information, it is indispensable in debugging problems with `mod_rewrite` configuration, since it will tell you exactly how each rule is processed.
mod_rewrite uses the Perl Compatible Regular Expression vocabulary. In this document, we do not attempt to provide a detailed reference to regular expressions. For that, we recommend the PCRE man pages, the Perl regular expression man page, and Mastering Regular Expressions, by Jeffrey Friedl.

In this document, we attempt to provide enough of a regex vocabulary to get you started, without being overwhelming, in the hope that RewriteRules will be scientific formulae, rather than magical incantations.

**Regex vocabulary**

The following are the minimal building blocks you will need, in order to write regular expressions and RewriteRules. They certainly do not represent a complete regular expression vocabulary, but they are a good place to start, and should help you read basic regular expressions, as well as write your own.

<table>
<thead>
<tr>
<th>Character</th>
<th>Meaning</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>.</td>
<td>Matches any single character</td>
<td>c.t will match cat, cot, cut, etc.</td>
</tr>
<tr>
<td>+</td>
<td>Repeats the previous match one or more times</td>
<td>a+ matches a, aa, aaa, etc</td>
</tr>
<tr>
<td>*</td>
<td>Repeats the previous match zero or more times</td>
<td>a* matches all the same things a+ matches, but will also match an empty string.</td>
</tr>
<tr>
<td>?</td>
<td>Makes the match optional.</td>
<td>colou?r will match color and colour.</td>
</tr>
<tr>
<td>^</td>
<td>Called an anchor, matches the beginning of the string</td>
<td>^a matches a string that begins with a</td>
</tr>
<tr>
<td>Symbol</td>
<td>Explanation</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>$</td>
<td>The other anchor, this matches the end of the string. a$ matches a string that ends with a.</td>
<td></td>
</tr>
<tr>
<td>( )</td>
<td>Groups several characters into a single unit, and captures a match for use in a backreference. (ab)+ matches ababab - that is, the + applies to the group. For more on backreferences see below.</td>
<td></td>
</tr>
<tr>
<td>[ ]</td>
<td>A character class - matches one of the characters c[ua]t matches cut, cot or cat.</td>
<td></td>
</tr>
<tr>
<td>[^ ]</td>
<td>Negative character class - matches any character not specified c[^/]t matches cat or c=t but not c/t</td>
<td></td>
</tr>
</tbody>
</table>

In **mod_rewrite** the ! character can be used before a regular expression to negate it. This is, a string will be considered to have matched only if it does not match the rest of the expression.

**Regex Back-Reference Availability**

One important thing here has to be remembered: Whenever you use parentheses in **Pattern** or in one of the **CondPattern**, back-references are internally created which can be used with the strings $N and %N (see below). These are available for creating the **Substitution** parameter of a **RewriteRule** or the **TestString** parameter of a **RewriteCond**.

Captures in the **RewriteRule** patterns are (counterintuitively) available to all preceding **RewriteCond** directives, because the **RewriteRule** expression is evaluated before the individual conditions.

Figure 1 shows to which locations the back-references are
transferred for expansion as well as illustrating the flow of the RewriteRule, RewriteCond matching. In the next chapters, we will be exploring how to use these back-references, so do not fret if it seems a bit alien to you at first.

Figure 1: The back-reference flow through a rule. In this example, a request for /test/1234 would be transformed into /admin.foo?page=test&id=1234&host=admin.example.com.
A RewriteRule consists of three arguments separated by spaces. The arguments are

1. Pattern: which incoming URLs should be affected by the rule;
2. Substitution: where should the matching requests be sent;
3. [flags]: options affecting the rewritten request.

The Pattern is a regular expression. It is initially (for the first rewrite rule or until a substitution occurs) matched against the URL-path of the incoming request (the part after the hostname but before any question mark indicating the beginning of a query string) or, in per-directory context, against the request’s path relative to the directory for which the rule is defined. Once a substitution has occurred, the rules that follow are matched against the substituted value.

The Substitution can itself be one of three things:

A full filesystem path to a resource
This maps a request to an arbitrary location on your filesystem, much like the `Alias` directive.

**A web-path to a resource**

```
RewriteRule "^/foo$" "/bar"
```

If `DocumentRoot` is set to `/usr/local/apache2/htdocs`, then this directive would map requests for `http://example.com/foo` to the path `/usr/local/apache2/htdocs/bar`.

**An absolute URL**

```
RewriteRule "^/product/view$" "http://site2.example.com/seeproduct.html"
```

This tells the client to make a new request for the specified URL.

The `Substitution` can also contain back-references to parts of the incoming URL-path matched by the `Pattern`. Consider the following:

```
RewriteRule "^/product/(.*)/view$" "/var/web/productdb/$1"
```

The variable $1 will be replaced with whatever text was matched by the expression inside the parenthesis in the `Pattern`. For example, a request for `http://example.com/product/r14df/view` will be mapped
to the path /var/web/productdb/r14df.

If there is more than one expression in parenthesis, they are available in order in the variables $1$, $2$, $3$, and so on.
The behavior of a RewriteRule can be modified by the application of one or more flags to the end of the rule. For example, the matching behavior of a rule can be made case-insensitive by the application of the [NC] flag:

```
RewriteRule "^puppy.html" "smalldog.html" [NC]
```

For more details on the available flags, their meanings, and examples, see the Rewrite Flags document.
One or more RewriteCond directives can be used to restrict the types of requests that will be subject to the following RewriteRule. The first argument is a variable describing a characteristic of the request, the second argument is a regular expression that must match the variable, and a third optional argument is a list of flags that modify how the match is evaluated.

![Syntax of a RewriteCond:](image)

**Figure 3: Syntax of the RewriteCond directive**

For example, to send all requests from a particular IP range to a different server, you could use:

```
RewriteCond %{REMOTE_ADDR} '^10\/.2\.'
RewriteRule "(.*)" "http://intranet.example.com$1"
```

When more than one RewriteCond is specified, they must all match for the RewriteRule to be applied. For example, to deny requests that contain the word "hack" in their query string, unless they also contain a cookie containing the word "go", you could use:

```
RewriteCond %{QUERY_STRING} "hack" [NC]
RewriteCond %{HTTP_COOKIE} "go" [NC]
RewriteRule "(.*)" "http://denied.example.com$1"
```
Notice that the exclamation mark specifies a negative match, so the rule is only applied if the cookie does not contain "go".

Matches in the regular expressions contained in the *RewriteCond* s can be used as part of the *Substitution* in the *RewriteRule* using the variables %1, %2, etc. For example, this will direct the request to a different directory depending on the hostname used to access the site:

```
RewriteCond "%{HTTP_HOST}" "(.*)"
RewriteRule "^/(.*)" "sites/%1/$1"
```

If the request was for `http://example.com/foo/bar`, then %1 would contain `example.com` and $1 would contain `foo/bar`. 
The RewriteMap directive provides a way to call an external function, so to speak, to do your rewriting for you. This is discussed in greater detail in the RewriteMap supplementary documentation.
Rewriting is typically configured in the main server configuration setting (outside any `<Directory>` section) or inside `<VirtualHost>` containers. This is the easiest way to do rewriting and is recommended. It is possible, however, to do rewriting inside `<Directory>` sections or `.htaccess` files at the expense of some additional complexity. This technique is called per-directory rewrites.

The main difference with per-server rewrites is that the path prefix of the directory containing the `.htaccess` file is stripped before matching in the `RewriteRule`. In addition, the `RewriteBase` should be used to assure the request is properly mapped.
RewriteRule Flags

This document discusses the flags which are available to the RewriteRule directive, providing detailed explanations and examples.

See also

- Module documentation
- mod_rewrite introduction
- Redirection and remapping
- Controlling access
- Virtual hosts
- Proxying
- Using RewriteMap
- Advanced techniques
- When not to use mod_rewrite
A RewriteRule can have its behavior modified by one or more flags. Flags are included in square brackets at the end of the rule, and multiple flags are separated by commas.

```plaintext
RewriteRule pattern target [Flag1,Flag2,Flag3]
```

Each flag (with a few exceptions) has a short form, such as CO, as well as a longer form, such as cookie. While it is most common to use the short form, it is recommended that you familiarize yourself with the long form, so that you remember what each flag is supposed to do. Some flags take one or more arguments. Flags are not case sensitive.

Flags that alter metadata associated with the request (T=, H=, E=) have no affect in per-directory and htaccess context, when a substitution (other than '-') is performed during the same round of rewrite processing.

Presented here are each of the available flags, along with an example of how you might use them.
The [B] flag instructs `RewriteRule` to escape non-alphanumeric characters before applying the transformation.

In 2.4.26 and later, you can limit the escaping to specific characters in backreferences by listing them: [B=#?;]. Note: The space character can be used in the list of characters to escape, but it cannot be the last character in the list.

`mod_rewrite` has to unescape URLs before mapping them, so backreferences are unescaped at the time they are applied. Using the B flag, non-alphanumeric characters in backreferences will be escaped. For example, consider the rule:

```
RewriteRule "^search/(.*)$" "/search.php?term=$1"
```

Given a search term of 'x & y/z', a browser will encode it as 'x%20%26%20y%2Fz', making the request 'search/x%20%26%20y%2Fz'. Without the B flag, this rewrite rule will map to 'search.php?term=x & y/z', which isn't a valid URL, and so would be encoded as `search.php?term=x%20&y%2Fz=`, which is not what was intended.

With the B flag set on this same rule, the parameters are re-encoded before being passed on to the output URL, resulting in a correct mapping to `/search.php?term=x%20%26%20y%2Fz`.

```
RewriteRule "^search/(.*)$" "/search.php?term=$1"
```

Note that you may also need to set `AllowEncodedSlashes` to On to get this particular example to work, as httpd does not allow encoded slashes in URLs, and returns a 404 if it sees one.
This escaping is particularly necessary in a proxy situation, when the backend may break if presented with an unescaped URL.

An alternative to this flag is using a `RewriteCond` to capture against `%{THE_REQUEST}` which will capture strings in the encoded form.
The [BNP] flag instructs RewriteRule to escape the space character in a backreference to %20 rather than '+'. Useful when the backreference will be used in the path component rather than the query string.

This flag is available in version 2.4.26 and later.
The [C] or [chain] flag indicates that the RewriteRule is chained to the next rule. That is, if the rule matches, then it is processed as usual and control moves on to the next rule. However, if it does not match, then the next rule, and any other rules that are chained together, are skipped.
The [CO] or [cookie] flag, allows you to set a cookie when a particular RewriteRule matches. The argument consists of three required fields and four optional fields.

The full syntax for the flag, including all attributes, is as follows:


If a literal `:` character is needed in any of the cookie fields, an alternate syntax is available. To opt-in to the alternate syntax, the cookie "Name" should be preceded with a `;` character, and field separators should be specified as `;`.

[CO=;NAME;VALUE:MOREVALUE:DOMAIN;lifetime:path;secure;httponly]

You must declare a name, a value, and a domain for the cookie to be set.

**Domain**

The domain for which you want the cookie to be valid. This may be a hostname, such as www.example.com, or it may be a domain, such as .example.com. It must be at least two parts separated by a dot. That is, it may not be merely .com or .net. Cookies of that kind are forbidden by the cookie security model.

You may optionally also set the following values:

**Lifetime**

The time for which the cookie will persist, in minutes. A value of 0 indicates that the cookie will persist only for the current browser session. This is the default value if none is specified.
**Path**

The path, on the current website, for which the cookie is valid, such as /customers/ or /files/download/.
By default, this is set to / - that is, the entire website.

**Secure**

If set to secure, true, or 1, the cookie will only be permitted to be translated via secure (https) connections.

**HttpOnly**

If set to HttpOnly, true, or 1, the cookie will have the HttpOnly flag set, which means that the cookie is inaccessible to JavaScript code on browsers that support this feature.

Consider this example:

```
RewriteEngine On
RewriteRule "^/index.html" "-" [CO=frontdoor:yes:.example.com:1440:/]
```

In the example given, the rule doesn't rewrite the request. The "-" rewrite target tells mod_rewrite to pass the request through unchanged. Instead, it sets a cookie called 'frontdoor' to a value of 'yes'. The cookie is valid for any host in the .example.com domain. It is set to expire in 1440 minutes (24 hours) and is returned for all URIs.
The DPI flag causes the PATH_INFO portion of the rewritten URI to be discarded.

This flag is available in version 2.2.12 and later.

In per-directory context, the URI each `RewriteRule` compares against is the concatenation of the current values of the URI and PATH_INFO.

The current URI can be the initial URI as requested by the client, the result of a previous round of mod_rewrite processing, or the result of a prior rule in the current round of mod_rewrite processing.

In contrast, the PATH_INFO that is appended to the URI before each rule reflects only the value of PATH_INFO before this round of mod_rewrite processing. As a consequence, if large portions of the URI are matched and copied into a substitution in multiple `RewriteRule` directives, without regard for which parts of the URI came from the current PATH_INFO, the final URI may have multiple copies of PATH_INFO appended to it.

Use this flag on any substitution where the PATH_INFO that resulted from the previous mapping of this request to the filesystem is not of interest. This flag permanently forgets the PATH_INFO established before this round of mod_rewrite processing began. PATH_INFO will not be recalculated until the current round of mod_rewrite processing completes. Subsequent rules during this round of processing will see only the direct result of substitutions, without any PATH_INFO appended.
With the [E], or [env] flag, you can set the value of an environment variable. Note that some environment variables may be set after the rule is run, thus unsetting what you have set. See the Environment Variables document for more details on how Environment variables work.

The full syntax for this flag is:

```
[E=VAR:VAL ]
[E=!VAR ]
```

VAL may contain backreferences ($N or %N) which are expanded.

Using the short form

```
[E=VAR ]
```

you can set the environment variable named VAR to an empty value.

The form

```
[E=!VAR ]
```

allows to unset a previously set environment variable named VAR.

Environment variables can then be used in a variety of contexts, including CGI programs, other RewriteRule directives, or CustomLog directives.

The following example sets an environment variable called 'image' to a value of '1' if the requested URI is an image file. Then, that environment variable is used to exclude those requests from the access log.
Note that this same effect can be obtained using `SetEnvIf`. This technique is offered as an example, not as a recommendation.
Using the [END] flag terminates not only the current round of rewrite processing (like [L]) but also prevents any subsequent rewrite processing from occurring in per-directory (htaccess) context.

This does not apply to new requests resulting from external redirects.
Using the [F] flag causes the server to return a 403 Forbidden status code to the client. While the same behavior can be accomplished using the Deny directive, this allows more flexibility in assigning a Forbidden status.

The following rule will forbid .exe files from being downloaded from your server.

```
RewriteRule "\.exe" "-" [F]
```

This example uses the "-" syntax for the rewrite target, which means that the requested URI is not modified. There's no reason to rewrite to another URI, if you're going to forbid the request.

When using [F], an [L] is implied - that is, the response is returned immediately, and no further rules are evaluated.
The [G] flag forces the server to return a 410 Gone status with the response. This indicates that a resource used to be available, but is no longer available.

As with the [F] flag, you will typically use the "-" syntax for the rewrite target when using the [G] flag:

```
RewriteRule "oldproduct" "-" [G,NC]
```

When using [G], an [L] is implied - that is, the response is returned immediately, and no further rules are evaluated.
Forces the resulting request to be handled with the specified handler. For example, one might use this to force all files without a file extension to be parsed by the php handler:

```
RewriteRule "!\." "-" [H=application/x-httpd-php]
```

The regular expression above - !\. - will match any request that does not contain the literal . character.

This can be also used to force the handler based on some conditions. For example, the following snippet used in per-server context allows .php files to be displayed by mod_php if they are requested with the .phps extension:

```
RewriteRule "^(/source/.+\.php)s$" "$1" [H=application/x-httpd-php-source]
```

The regular expression above - ^(/source/.+\.php)s$ - will match any request that starts with /source/ followed by 1 or n characters followed by .phps literally. The backreference $1 referrers to the captured match within parenthesis of the regular expression.
The [L] flag causes mod rewrite to stop processing the rule set. In most contexts, this means that if the rule matches, no further rules will be processed. This corresponds to the last command in Perl, or the break command in C. Use this flag to indicate that the current rule should be applied immediately without considering further rules.

If you are using RewriteRule in either .htaccess files or in <Directory> sections, it is important to have some understanding of how the rules are processed. The simplified form of this is that once the rules have been processed, the rewritten request is handed back to the URL parsing engine to do what it may with it. It is possible that as the rewritten request is handled, the .htaccess file or <Directory> section may be encountered again, and thus the ruleset may be run again from the start. Most commonly this will happen if one of the rules causes a redirect - either internal or external - causing the request process to start over.

It is therefore important, if you are using RewriteRule directives in one of these contexts, that you take explicit steps to avoid rules looping, and not count solely on the [L] flag to terminate execution of a series of rules, as shown below.

An alternative flag, [END], can be used to terminate not only the current round of rewrite processing but prevent any subsequent rewrite processing from occurring in per-directory (htaccess) context. This does not apply to new requests resulting from external redirects.

The example given here will rewrite any request to index.php, giving the original request as a query string argument to index.php, however, the RewriteCond ensures that if the
request is already for index.php, the **RewriteRule** will be skipped.

```
RewriteBase "/
RewriteCond "%{REQUEST_URI}" "!=/index.php"
RewriteRule "^(.*)" "index.php?req=$1" [L,]
```
The [N] flag causes the ruleset to start over again from the top, using the result of the ruleset so far as a starting point. Use with extreme caution, as it may result in loop.

The [Next] flag could be used, for example, if you wished to replace a certain string or letter repeatedly in a request. The example shown here will replace A with B everywhere in a request, and will continue doing so until there are no more As to be replaced.

```
RewriteRule "(.*)A(.*))" "$1B$2" [N]
```

You can think of this as a while loop: While this pattern still matches (i.e., while the URI still contains an A), perform this substitution (i.e., replace the A with a B).

In 2.4.8 and later, this module returns an error after 32,000 iterations to protect against unintended looping. An alternative maximum number of iterations can be specified by adding to the N flag.

```
# Be willing to replace 1 character in each
RewriteRule "(.+)\[><;\]$" "$1" [N=64000]
# ... or, give up if after 10 loops
RewriteRule "(.+)\[><;\]$" "$1" [N=10]
```
Use of the [NC] flag causes the RewriteRule to be matched in a case-insensitive manner. That is, it doesn't care whether letters appear as upper-case or lower-case in the matched URI.

In the example below, any request for an image file will be proxied to your dedicated image server. The match is case-insensitive, so that .jpg and .JPG files are both acceptable, for example.

```
RewriteRule "(.*/.(jpg|gif|png))$" "http://images.example.com$1"
```
By default, special characters, such as & and ?, for example, will be converted to their hexcode equivalent. Using the [NE] flag prevents that from happening.

```
RewriteRule "^/anchor/(.+)" "/bigpage.html#$1"
```

The above example will redirect /anchor/xyz to /bigpage.html#xyz. Omitting the [NE] will result in the # being converted to its hexcode equivalent, %23, which will then result in a 404 Not Found error condition.
Use of the [NS] flag prevents the rule from being used on subrequests. For example, a page which is included using an SSI (Server Side Include) is a subrequest, and you may want to avoid rewrites happening on those subrequests. Also, when `mod_dir` tries to find out information about possible directory default files (such as `index.html` files), this is an internal subrequest, and you often want to avoid rewrites on such subrequests. On subrequests, it is not always useful, and can even cause errors, if the complete set of rules are applied. Use this flag to exclude problematic rules.

To decide whether or not to use this rule: if you prefix URLs with CGI-scripts, to force them to be processed by the CGI-script, it's likely that you will run into problems (or significant overhead) on sub-requests. In these cases, use this flag.

Images, javascript files, or css files, loaded as part of an HTML page, are not subrequests - the browser requests them as separate HTTP requests.
Use of the [P] flag causes the request to be handled by
mod_proxy, and handled via a proxy request. For example, if you
wanted all image requests to be handled by a back-end image
server, you might do something like the following:

```
RewriteRule "/(.*)\.(jpg|gif|png)" "http://
```

Use of the [P] flag implies [L] - that is, the request is immediately
pushed through the proxy, and any following rules will not be
considered.

You must make sure that the substitution string is a valid URI
(typically starting with http://hostname) which can be handled
by the mod_proxy. If not, you will get an error from the proxy
module. Use this flag to achieve a more powerful implementation
of the ProxyPass directive, to map remote content into the
namespace of the local server.

Security Warning

Take care when constructing the target URL of the rule,
considering the security impact from allowing the client
influence over the set of URLs to which your server will act as a
proxy. Ensure that the scheme and hostname part of the URL is
either fixed, or does not allow the client undue influence.

Performance warning

Using this flag triggers the use of mod_proxy, without handling
of persistent connections. This means the performance of your
proxy will be better if you set it up with ProxyPass or
ProxyPassMatch
This is because this flag triggers the use of the default worker, which does not handle connection pooling/reuse. Avoid using this flag and prefer those directives, whenever you can.

Note: `mod_proxy` must be enabled in order to use this flag.
The target (or substitution string) in a RewriteRule is assumed to be a file path, by default. The use of the [PT] flag causes it to be treated as a URI instead. That is to say, the use of the [PT] flag causes the result of the RewriteRule to be passed back through URL mapping, so that location-based mappings, such as Alias, Redirect, or ScriptAlias, for example, might have a chance to take effect.

If, for example, you have an Alias for /icons, and have a RewriteRule pointing there, you should use the [PT] flag to ensure that the Alias is evaluated.

```
Alias "/icons" "/usr/local/apache/icons"
RewriteRule "/pics/(.+).jpg$" "/icons/$1.gif"
```

Omission of the [PT] flag in this case will cause the Alias to be ignored, resulting in a 'File not found' error being returned.

The PT flag implies the L flag: rewriting will be stopped in order to pass the request to the next phase of processing.

Note that the PT flag is implied in per-directory contexts such as `<Directory>` sections or in .htaccess files. The only way to circumvent that is to rewrite to `-.`
When the replacement URI contains a query string, the default behavior of RewriteRule is to discard the existing query string, and replace it with the newly generated one. Using the [QSA] flag causes the query strings to be combined.

Consider the following rule:

```
RewriteRule "/pages/(.+)/" "/page.php?page=$1"
```

With the [QSA] flag, a request for /pages/123?one=two will be mapped to /page.php?page=123&one=two. Without the [QSA] flag, that same request will be mapped to /page.php?page=123 - that is, the existing query string will be discarded.
When the requested URI contains a query string, and the target URI does not, the default behavior of `RewriteRule` is to copy that query string to the target URI. Using the [QSD] flag causes the query string to be discarded.

This flag is available in version 2.4.0 and later.

Using [QSD] and [QSA] together will result in [QSD] taking precedence.

If the target URI has a query string, the default behavior will be observed - that is, the original query string will be discarded and replaced with the query string in the `RewriteRule` target URI.
By default, the first (left-most) question mark in the substitution delimits the path from the query string. Using the [QSL] flag instructs `RewriteRule` to instead split the two components using the last (right-most) question mark.

This is useful when mapping to files that have literal question marks in their filename. If no query string is used in the substitution, a question mark can be appended to it in combination with this flag.

This flag is available in version 2.4.19 and later.
Use of the [R] flag causes a HTTP redirect to be issued to the browser. If a fully-qualified URL is specified (that is, including http://servername/) then a redirect will be issued to that location. Otherwise, the current protocol, servername, and port number will be used to generate the URL sent with the redirect.

Any valid HTTP response status code may be specified, using the syntax [R=305], with a 302 status code being used by default if none is specified. The status code specified need not necessarily be a redirect (3xx) status code. However, if a status code is outside the redirect range (300-399) then the substitution string is dropped entirely, and rewriting is stopped as if the L were used.

In addition to response status codes, you may also specify redirect status using their symbolic names: temp (default), permanent, or seeother.

You will almost always want to use [R] in conjunction with [L] (that is, use [R,L]) because on its own, the [R] flag prepends http://thishost[:thisport] to the URI, but then passes this on to the next rule in the ruleset, which can often result in 'Invalid URI in request' warnings.
The [S] flag is used to skip rules that you don't want to run. The syntax of the skip flag is [S=N], where N signifies the number of rules to skip (provided the RewriteRule matches). This can be thought of as a goto statement in your rewrite ruleset. In the following example, we only want to run the RewriteRule if the requested URI doesn't correspond with an actual file.

```plaintext
# Is the request for a non-existent file?
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# If so, skip these two RewriteRules
RewriteRule .? - [S=2]
RewriteRule ".*.gif" "images.php?$1"
RewriteRule ".*.html" "docs.php?$1"
```

This technique is useful because a RewriteCond only applies to the RewriteRule immediately following it. Thus, if you want to make a RewriteCond apply to several RewriteRules, one possible technique is to negate those conditions and add a RewriteRule with a [Skip] flag. You can use this to make pseudo if-then-else constructs: The last rule of the then-clause becomes skip=N, where N is the number of rules in the else-clause:

```plaintext
# Does the file exist?
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
# Create an if-then-else construct by skipping 3 lines if we meant to go to the "else" stanza.
RewriteRule .? - [S=3]

# IF the file exists, then:
    RewriteRule ".*.gif" "images.php?$1"
    RewriteRule ".*.html" "docs.php?$1"
# Skip past the "else" stanza.
```
It is probably easier to accomplish this kind of configuration using the `<If>`, `<ElseIf>`, and `<Else>` directives instead.
Sets the MIME type with which the resulting response will be sent. This has the same effect as the `AddType` directive.

For example, you might use the following technique to serve Perl source code as plain text, if requested in a particular way:

```
# Serve .pl files as plain text
RewriteRule ".\.pl$" "-" [T=text/plain]
```

Or, perhaps, if you have a camera that produces jpeg images without file extensions, you could force those images to be served with the correct MIME type by virtue of their file names:

```
# Files with 'IMG' in the name are jpg images
RewriteRule "IMG" "-" [T=image/jpg]
```

Please note that this is a trivial example, and could be better done using `<FilesMatch>` instead. Always consider the alternate solutions to a problem before resorting to rewrite, which will invariably be a less efficient solution than the alternatives.

If used in per-directory context, use only - (dash) as the substitution for the entire round of mod_rewrite processing, otherwise the MIME-type set with this flag is lost due to an internal re-processing (including subsequent rounds of mod_rewrite processing). The L flag can be useful in this context to end the current round of mod_rewrite processing.
Developing modules for the Apache HTTP Server 2.4

This document explains how you can develop modules for the Apache HTTP Server 2.4

See also

Request Processing in Apache 2.4
Apache 2.x Hook Functions
What we will be discussing in this document

This document will discuss how you can create modules for the Apache HTTP Server 2.4, by exploring an example module called mod_example. In the first part of this document, the purpose of this module will be to calculate and print out various digest values for existing files on your web server, whenever we access the URL http://hostname/filename.sum. For instance, if we want to know the MD5 digest value of the file located at http://www.example.com/index.html, we would visit http://www.example.com/index.html.sum.

In the second part of this document, which deals with configuration directive and context awareness, we will be looking at a module that simply writes out its own configuration to the client.

Prerequisites

First and foremost, you are expected to have a basic knowledge of how the C programming language works. In most cases, we will try to be as pedagogical as possible and link to documents describing the functions used in the examples, but there are also many cases where it is necessary to either just assume that "it works" or do some digging yourself into what the hows and whys of various function calls.

Lastly, you will need to have a basic understanding of how modules are loaded and configured in the Apache HTTP Server, as well as how to get the headers for Apache if you do not have them already, as these are needed for compiling new modules.

Compiling your module

To compile the source code we are building in this document, we
will be using **APXS**. Assuming your source file is called mod_example.c, compiling, installing and activating the module is as simple as:

```bash
apxs -i -a -c mod_example.c
```
Every module starts with the same declaration, or name tag if you will, that defines a module as a separate entity within Apache:

```c
module AP_MODULE_DECLARE_DATA example_module {
    STANDARD20_MODULE_STUFF,
    create_dir_conf,  /* Per-directory configuration handler */
    merge_dir_conf,  /* Merge handler for per-directory configurations */
    create_svr_conf,  /* Per-server configuration handler */
    merge_svr_conf,  /* Merge handler for per-server configurations */
    directives,  /* Any directives we may have for httpd */
    register_hooks   /* Our hook registering function */
};
```

This bit of code lets the server know that we have now registered a new module in the system, and that its name is `example_module`. The name of the module is used primarily for two things:

- Letting the server know how to load the module using the `LoadModule` directive
- Setting up a namespace for the module to use in configurations

For now, we're only concerned with the first purpose of the module name, which comes into play when we need to load the module:

```bash
LoadModule example_module modules/mod_example.so
```
In essence, this tells the server to open up `mod_example.so` and look for a module called `example_module`.

Within this name tag of ours is also a bunch of references to how we would like to handle things: Which directives do we respond to in a configuration file or `.htaccess`, how do we operate within specific contexts, and what handlers are we interested in registering with the Apache HTTP service. We'll return to all these elements later in this document.
Getting started: Hooking into the server

An introduction to hooks

When handling requests in Apache HTTP Server 2.4, the first thing you will need to do is create a hook into the request handling process. A hook is essentially a message telling the server that you are willing to either serve or at least take a glance at certain requests given by clients. All handlers, whether it's mod_rewrite, mod_authn_*, mod_proxy and so on, are hooked into specific parts of the request process. As you are probably aware, modules serve different purposes; Some are authentication/authorization handlers, others are file or script handlers while some third modules rewrite URIs or proxies content. Furthermore, in the end, it is up to the user of the server how and when each module will come into place. Thus, the server itself does not presume to know which module is responsible for handling a specific request, and will ask each module whether they have an interest in a given request or not. It is then up to each module to either gently decline serving a request, accept serving it or flat out deny the request from being served, as authentication/authorization modules do:

To make it a bit easier for handlers such as our mod_example to know whether the client is requesting content we should handle or not, the server has directives for hinting to modules whether their
assistance is needed or not. Two of these are AddHandler and SetHandler. Let's take a look at an example using AddHandler. In our example case, we want every request ending with .sum to be served by mod_example, so we'll add a configuration directive that tells the server to do just that:

```
AddHandler example-handler .sum
```

What this tells the server is the following: Whenever we receive a request for a URI ending in .sum, we are to let all modules know that we are looking for whoever goes by the name of "example-handler". Thus, when a request is being served that ends in .sum, the server will let all modules know, that this request should be served by "example-handler ". As you will see later, when we start building mod_example, we will check for this handler tag relayed by AddHandler and reply to the server based on the value of this tag.

**Hooking into httpd**

To begin with, we only want to create a simple handler, that replies to the client browser when a specific URL is requested, so we won't bother setting up configuration handlers and directives just yet. Our initial module definition will look like this:

```c
module AP_MODULE_DECLARE_DATA example_module {

    STANDARD20_MODULE_STUFF,
    NULL,
    NULL,
    NULL,
    NULL,
    NULL,
    register_hooks    /* Our hook registering */
};
```
This lets the server know that we are not interested in anything fancy, we just want to hook onto the requests and possibly handle some of them.

The reference in our example declaration, `register_hooks` is the name of a function we will create to manage how we hook onto the request process. In this example module, the function has just one purpose; To create a simple hook that gets called after all the rewrites, access control etc has been handled. Thus, we will let the server know, that we want to hook into its process as one of the last modules:

```c
static void register_hooks(apr_pool_t *pool)
{
  /* Create a hook in the request handler
   * ap_hook_handler(example_handler, NULL, NULL, APR_HOOK_LAST);
  }
```

The `example_handler` reference is the function that will handle the request. We will discuss how to create a handler in the next chapter.

**Other useful hooks**

Hooking into the request handling phase is but one of many hooks that you can create. Some other ways of hooking are:

- `ap_hook_child_init`: Place a hook that executes when a child process is spawned (commonly used for initializing modules after the server has forked)
- `ap_hook_pre_config`: Place a hook that executes before any configuration data has been read (very early hook)
- **ap_hook_post_config**: Place a hook that executes after configuration has been parsed, but before the server has forked.

- **ap_hook_translate_name**: Place a hook that executes when a URI needs to be translated into a filename on the server (think `mod_rewrite`).

- **ap_hook_quick_handler**: Similar to `ap_hook_handler`, except it is run before any other request hooks (translation, auth, fixups etc).

- **ap_hook_log_transaction**: Place a hook that executes when the server is about to add a log entry of the current request.
A handler is essentially a function that receives a callback when a request to the server is made. It is passed a record of the current request (how it was made, which headers and requests were passed along, who's giving the request and so on), and is put in charge of either telling the server that it's not interested in the request or handle the request with the tools provided.

**A simple "Hello, world!" handler**

Let's start off by making a very simple request handler that does the following:

1. Check that this is a request that should be served by "example-handler"
2. Set the content type of our output to text/html
3. Write "Hello, world!" back to the client browser
4. Let the server know that we took care of this request and everything went fine

In C code, our example handler will now look like this:

```c
static int example_handler(request_rec *r) {
    /* First off, we need to check if this is a call for the "example-handler" handler. */
    if (!r->handler || strcmp(r->handler, "example-handler")) return DECLINED;

    /* Now that we are handling this request, we must first set the appropriate content type and output. */
    ap_set_content_type(r, "text/html");
    ap_rprintf(r, "Hello, world!");
}
```
Lastly, we must tell the server that we took care of this request and everything went fine. We do so by simply returning the value OK to the server.

```c
return OK;
```

Now, we put all we have learned together and end up with a program that looks like `mod_example_1.c`. The functions used in this example will be explained later in the section "Some useful functions you should know".

The request_rec structure

The most essential part of any request is the request record. In a call to a handler function, this is represented by the `request_rec*` structure passed along with every call that is made. This struct, typically just referred to as `r` in modules, contains all the information you need for your module to fully process any HTTP request and respond accordingly.

Some key elements of the `request_rec` structure are:

- `r->handler (char*)`: Contains the name of the handler the server is currently asking to do the handling of this request
- `r->method (char*)`: Contains the HTTP method being used, f.x. GET or POST
- `r->filename (char*)`: Contains the translated filename the client is requesting
- `r->args (char*)`: Contains the query string of the request, if any
- `r->headers_in (apr_table_t*)`: Contains all the headers sent by the client
- `r->connection (conn_rec*)`: A record containing
information about the current connection
- **r->user (char*)**: If the URI requires authentication, this is set to the username provided
- **r->useragent_ip (char*)**: The IP address of the client connecting to us
- **r->pool (apr_pool_t*)**: The memory pool of this request. We'll discuss this in the "Memory management" chapter.

A complete list of all the values contained within the request_rec structure can be found in the **httpd.h** header file or at [http://ci.apache.org/projects/httpd/trunk/doxygen/structrequest__rec.html](http://ci.apache.org/projects/httpd/trunk/doxygen/structrequest__rec.html)

Let's try out some of these variables in another example handler:

```c
static int example_handler(request_rec *r) {
    /* Set the appropriate content type */
    ap_set_content_type(r, "text/html");

    /* Print out the IP address of the client */
    ap_rprintf(r, "<h2>Hello, %s!</h2>", r->useragent_ip);

    /* If we were reached through a GET or a POST request */
    if ( !strcmp(r->method, "POST") || !strcmp(r->method, "GET") ) {
        ap_rputs("You used a GET or a POST method,
                that makes us happy!<br/>", r);
    } else {
        ap_rputs("You did not use POST or GET,
              that makes us sad :(<br/>", r);
    }

    /* Lastly, if there was a query string, */
    if (r->args) {
        ap_rprintf(r, "Your query string was:
              %s", r->args);
    }
}
```
Return values

Apache relies on return values from handlers to signify whether a request was handled or not, and if so, whether the request went well or not. If a module is not interested in handling a specific request, it should always return the value DECLINED. If it is handling a request, it should either return the generic value OK, or a specific HTTP status code, for example:

```c
static int example_handler(request_rec *r)
{
    /* Return 404: Not found */
    return HTTP_NOT_FOUND;
}
```

Returning OK or a HTTP status code does not necessarily mean that the request will end. The server may still have other handlers that are interested in this request, for instance the logging modules which, upon a successful request, will write down a summary of what was requested and how it went. To do a full stop and prevent any further processing after your module is done, you can return the value DONE to let the server know that it should cease all activity on this request and carry on with the next, without informing other handlers.

**General response codes:**

- DECLINED: We are not handling this request
- OK: We handled this request and it went well
- DONE: We handled this request and the server should just close this thread without further processing
HTTP specific return codes (excerpt):

- HTTP_OK (200): Request was okay
- HTTP_MOVED_PERMANENTLY (301): The resource has moved to a new URL
- HTTP_UNAUTHORIZED (401): Client is not authorized to visit this page
- HTTP_FORBIDDEN (403): Permission denied
- HTTP_NOT_FOUND (404): File not found
- HTTP_INTERNAL_SERVER_ERROR (500): Internal server error (self explanatory)

Some useful functions you should know

- `ap_rputs(const char *string, request_rec *r)`:
  Sends a string of text to the client. This is a shorthand version of `ap_rwrite`.

  ```c
  ap_rputs("Hello, world!", r);
  ```

- `ap_rprintf`:
  This function works just like printf, except it sends the result to the client.

  ```c
  ap_rprintf(r, "Hello, %s!", r->useragent);
  ```

- `ap_set_content_type(request_rec *r, const char *type)`:
  Sets the content type of the output you are sending.

  ```c
  ap_set_content_type(r, "text/plain"); /*
  ```
Memory management

Managing your resources in Apache HTTP Server 2.4 is quite easy, thanks to the memory pool system. In essence, each server, connection and request have their own memory pool that gets cleaned up when its scope ends, e.g. when a request is done or when a server process shuts down. All your module needs to do is latch onto this memory pool, and you won’t have to worry about having to clean up after yourself - pretty neat, huh?

In our module, we will primarily be allocating memory for each request, so it’s appropriate to use the r->pool reference when creating new objects. A few of the functions for allocating memory within a pool are:

- void* apr_palloc( apr_pool_t *p, apr_size_t size): Allocates size number of bytes in the pool for you
- void* apr_pmalloc( apr_pool_t *p, apr_size_t size): Allocates size number of bytes in the pool for you and sets all bytes to 0
- char* apr_pstrdup( apr_pool_t *p, const char *s): Creates a duplicate of the string s. This is useful for copying constant values so you can edit them
- char* apr_psprintf( apr_pool_t *p, const char *fmt, ...): Similar to sprintf, except the server supplies you with an appropriately allocated target variable

Let’s put these functions into an example handler:

```c
static int example_handler(request_rec *r) {
    const char *original = "You can't edit this!"
    char *copy;
    int *integers;
```
/* Allocate space for 10 integer values */
integers = apr_pcalloc(r->pool, sizeof(int) * 10);

/* Create a copy of the 'original' variable */
copy = apr_pstrdup(r->pool, original);
return OK;

This is all well and good for our module, which won't need any pre-initialized variables or structures. However, if we wanted to initialize something early on, before the requests come rolling in, we could simply add a call to a function in our register_hooks function to sort it out:

static void register_hooks(apr_pool_t *pool)
{
    /* Call a function that initializes some */
    example_init_function(pool);
    /* Create a hook in the request handler */
    ap_hook_handler(example_handler, NULL, NULL, APR_HOOK_LAST);
}

In this pre-request initialization function we would not be using the same pool as we did when allocating resources for request-based functions. Instead, we would use the pool given to us by the server for allocating memory on a per-process based level.

**Parsing request data**

In our example module, we would like to add a feature, that checks which type of digest, MD5 or SHA1 the client would like to see. This could be solved by adding a query string to the request. A query string is typically comprised of several keys and values put together in a string, for instance
valueA=yes&valueB=no&valueC=maybe. It is up to the module itself to parse these and get the data it requires. In our example, we'll be looking for a key called digest, and if set to md5, we'll produce an MD5 digest, otherwise we'll produce a SHA1 digest.

Since the introduction of Apache HTTP Server 2.4, parsing request data from GET and POST requests have never been easier. All we require to parse both GET and POST data is four simple lines:

```c
apr_table_t *GET;
apr_array_header_t*POST;
ap_args_to_table(r, &GET);
ap_parse_form_data(r, NULL, &POST, -1, 8192);
```

In our specific example module, we're looking for the digest value from the query string, which now resides inside a table called GET. To extract this value, we need only perform a simple operation:

```c
/* Get the "digest" key from the query string */
const char *digestType = apr_table_get(GET,
/* If no key was returned, we will set a default */
if (!digestType) digestType = "sha1";
```

The structures used for the POST and GET data are not exactly the same, so if we were to fetch a value from POST data instead
of the query string, we would have to resort to a few more lines, as outlined in this example in the last chapter of this document.

**Making an advanced handler**

Now that we have learned how to parse form data and manage our resources, we can move on to creating an advanced version of our module, that spits out the MD5 or SHA1 digest of files:

```c
static int example_handler(request_rec *r) {
    int rc, exists;
    apr_finfo_t finfo;
    apr_file_t *file;
    char *filename;
    char buffer[256];
    apr_size_t readBytes;
    int n;
    apr_table_t *GET;
    apr_array_header_t *POST;
    const char *digestType;

    /* Check that the "example-handler" handler is being called. */
    if (!r->handler || strcmp(r->handler, "example-handler")) return (DECLINED);

    /* Figure out which file is being requested by removing the .sum from it. */
    filename = apr_pstrdup(r->pool, r->filename);
    filename[strlen(filename)-4] = 0; /* Cut off the last 4 characters. */

    /* Figure out if the file we request actually exists and isn't a directory. */
    rc = apr_stat(&finfo, filename, APR_FINFO_MIN, r->pool);
    if (rc == APR_SUCCESS) {
        exists =
        (finfo.filetype != APR_NOFILE)
        && !(finfo.filetype & APR_DIR)
```
if (!exists) return HTTP_NOT_FOUND;
/*
Return a 404 if not found.
*/

/*
If apr_stat failed, we're probably not allowed to check this file.
*/
else return HTTP_FORBIDDEN;

/* Parse the GET and, optionally, the POST data sent to us */
ap_args_to_table(r, &GET);
ap_parse_form_data(r, NULL, &POST, -1, -1);

/* Set the appropriate content type */
ap_set_content_type(r, "text/html");

/* Print a title and some general information */
ap_rprintf(r, "<h2>Information on %s: </h2>", filename);
ap_rprintf(r, "<b>Size: </b>%u bytes<br/>");

/* Get the digest type the client wants */
digestType = apr_table_get(GET, "digest");
if (!digestType) digestType = "MD5";

rc = apr_file_open(&file, filename, APR_READ, APR_OS_DEFAULT, r->pool);
if (rc == APR_SUCCESS) {
    /* Are we trying to calculate the MD5? */
    if (!strcasecmp(digestType, "md5"))
        /* Calculate the MD5 sum of the file */
        union {
            char chr[16];
            uint32_t num[4];
        } digest;
        apr_md5_ctx_t md5;
        apr_md5_init(&md5);
        readBytes_init(&md5);
        readBytes = 256;
        while (apr_file_read(file, buffer, &readBytes) == APR_SUCCESS)
            apr_md5_update(&md5, buffer, readBytes);
        apr_md5_getdigest(&md5, &digest.chr, &digest.num);
        autorelease("md5 = "); ap_hex_string(r, &digest.chr, 16);
    else
        /* Calculate the SHA1 sum of the file */
        Apr_sha1_ctx_t sha1;
        apr_sha1_init(&sha1);
        readBytes_init(&sha1);
        readBytes = 256;
        while (apr_file_read(file, buffer, &readBytes) == APR_SUCCESS)
            apr_sha1_update(&sha1, buffer, readBytes);
        apr_sha1_getdigest(&sha1, &digest.chr, &digest.num);
apr_md5_update(&md5, buffer);
apr_md5_final(digest.chr, &md5);

/* Print out the MD5 digest */
ap_rputs("<b>MD5: </b><code>", r);
for (n = 0; n < APR_MD5_DIGESTSIZE/4; n++)
ap_rprintf(r, "%08x", digest.num[n]);
ap_rputs("</code>", r);
/* Print a link to the SHA1 version */
ap_rputs("<br/><a href='?digest=sha1'>View the SHA1 hash instead</a>", r);
}
else {
/* Calculate the SHA1 sum of the file */
union {
    char chr[20];
    uint32_t num[5];
} digest;
apr_sha1_ctx_t sha1;
apr_sha1_init(&sha1);
readBytes = 256;
while (apr_file_read(file, buffer, &readBytes)) {
apr_sha1_update(&sha1, buffer);
}
apr_sha1_final(digest.chr, &sha1);

/* Print out the SHA1 digest */
ap_rputs("<b>SHA1: </b><code>", r);
for (n = 0; n < APR_SHA1_DIGESTSIZE/4; n++)
ap_rprintf(r, "%08x", digest.num[n]);
ap_rputs("</code>", r);
/* Print a link to the MD5 version */
ap_rputs("<br/><a href='?digest=md5'>View the MD5 hash instead</a>", r);
}
apr_file_close(file);

} /* Let the server know that we responded */
return OK;

This version in its entirety can be found here: mod_example_2.c.
In this next segment of this document, we will turn our eyes away from the digest module and create a new example module, whose only function is to write out its own configuration. The purpose of this is to examine how the server works with configuration, and what happens when you start writing advanced configurations for your modules.

**An introduction to configuration directives**

If you are reading this, then you probably already know what a configuration directive is. Simply put, a directive is a way of telling an individual module (or a set of modules) how to behave, such as these directives control how mod_rewrite works:

```plaintext
RewriteEngine On
RewriteCond %{REQUEST_URI} '^/foo/bar'
RewriteRule '^/foo/bar/(.*)$' '/foobar?page=$1'
```

Each of these configuration directives are handled by a separate function, that parses the parameters given and sets up a configuration accordingly.

**Making an example configuration**

To begin with, we'll create a basic configuration in C-space:

```c
typedef struct {
    int enabled;  /* Enable or disable our module */
    const char *path;  /* Some path to...something */
    int typeOfAction; /* 1 means action A, 2 means action B and so on */
} example_config;
```

Now, let's put this into perspective by creating a very small module
that just prints out a hard-coded configuration. You'll notice that we use the register_hooks function for initializing the configuration values to their defaults:

typedef struct {
    int enabled;       /* Enable or disable our module */
    const char *path;  /* Some path to...something */
    int typeOfAction;  /* 1 means action A, 2 means action B and so on */
} example_config;

static example_config config;

static int example_handler(request_rec *r) {
    if (!r->handler || strcmp(r->handler, "example-handler"))
        return(DECLINED);
    ap_set_content_type(r, "text/plain");
    ap_rprintf(r, "Enabled: %u\n", config.enabled);
    ap_rprintf(r, "Path: %s\n", config.path);
    ap_rprintf(r, "TypeOfAction: %x\n", config.typeOfAction);
    return OK;
}

static void register_hooks(apr_pool_t *pool) {
    config.enabled = 1;
    config.path = "/foo/bar";
    config.typeOfAction = 0x00;
    ap_hook_handler(example_handler, NULL, NULL, AP_HOOK_LAST);
}

/* Define our module as an entity and assign */
module AP_MODULE_DECLARE_DATA example_module {
    STANDARD20_MODULE_STUFF,
    NULL,  /* Per-directory configuration */
So far so good. To access our new handler, we could add the following to our configuration:

```
<Location "/example">
  SetHandler example-handler
</Location>
```

When we visit, we'll see our current configuration being spit out by our module.

**Registering directives with the server**

What if we want to change our configuration, not by hard-coding new values into the module, but by using either the httpd.conf file or possibly a .htaccess file? It's time to let the server know that we want this to be possible. To do so, we must first change our `name tag` to include a reference to the configuration directives we want to register with the server:

```
module AP_MODULE_DECLARE_DATA example_module {
  STANDARD_MODULE_STUFF,
  NULL, /* Per-directory configuration handler*/
  NULL, /* Merge handler for per-directory configurations*/
  NULL, /* Per-server configuration handler*/
  NULL, /* Merge handler for per-server configurations*/
  example_directives, /* Any directives we may have for httpd*/
  register_hooks /* Our hook registering function*/;
};
```
This will tell the server that we are now accepting directives from the configuration files, and that the structure called example_directives holds information on what our directives are and how they work. Since we have three different variables in our module configuration, we will add a structure with three directives and a NULL at the end:

```c
static const command_rec example_directives[] = {
    AP_INIT_TAKE1("exampleEnabled", example_set_enabled, NULL, RSRC_CONF, "Enable or disable mod_example"),
    AP_INIT_TAKE1("examplePath", example_set_path, NULL, RSRC_CONF, "The path to whatever"),
    AP_INIT_TAKE2("exampleAction", example_set_action, NULL, RSRC_CONF, "Special action value!")
};
```

As you can see, each directive needs at least 5 parameters set:

1. **AP_INIT_TAKE1**: This is a macro that tells the server that this directive takes one and only one argument. If we required two arguments, we could use the macro `AP_INIT_TAKE2` and so on (refer to httpd_conf.h for more macros).

2. `exampleEnabled`: This is the name of our directive. More precisely, it is what the user must put in his/her configuration in order to invoke a configuration change in our module.

3. `example_set_enabled`: This is a reference to a C function that parses the directive and sets the configuration
accordingly. We will discuss how to make this in the following paragraph.

4. **RSRC_CONF**: This tells the server where the directive is permitted. We'll go into details on this value in the later chapters, but for now, RSRC_CONF means that the server will only accept these directives in a server context.

5. "Enable or disable....": This is simply a brief description of what the directive does.

(\textit{The "missing" parameter in our definition, which is usually set to NULL, is an optional function that can be run after the initial function to parse the arguments have been run. This is usually omitted, as the function for verifying arguments might as well be used to set them.)}

**The directive handler function**

Now that we have told the server to expect some directives for our module, it's time to make a few functions for handling these. What the server reads in the configuration file(s) is text, and so naturally, what it passes along to our directive handler is one or more strings, that we ourselves need to recognize and act upon. You'll notice, that since we set our exampleAction directive to accept two arguments, its C function also has an additional parameter defined:

```c
/* Handler for the "exampleEnabled" directive */
const char *example_set_enabled(cmd_parms *cmd, void *cfg, const char *arg)
{
    if(!strcasecmp(arg, "on")) config.enabled = 1;
    else config.enabled = 0;
    return NULL;
}
```
Putting it all together

Now that we have our directives set up, and handlers configured for them, we can assemble our module into one big file:

```c
/* mod_example_config_simple.c: */
#include <stdio.h>
#include "apr_hash.h"
#include "ap_config.h"
#include "ap_provider.h"
#include "httpd.h"
#include "http_core.h"
#include "http_config.h"
#include "http_log.h"
#include "http_protocol.h"
#include "http_request.h"
```
typedef struct {
    int enabled; /* Enable or disable our module */
    const char *path; /* Some path to...something */
    int typeOfAction; /* 1 means action A, 2 means action B and so on */
} example_config;

static example_config config;

/*
 * Our directive handlers:
 */
/*
 * Handler for the "exampleEnabled" directive
 */
const char *example_set_enabled(cmd_parms *cmd, void *cfg, const char *arg)
{
    if(!strcasecmp(arg, "on")) config.enabled = 1;
    else config.enabled = 0;
    return NULL;
}

/*
 * Handler for the "examplePath" directive
 */
const char *example_set_path(cmd_parms *cmd)
{
    config.path = arg;
    return NULL;
}

/*
 * Handler for the "exampleAction" directive
 */
/*
 * Let's pretend this one takes one argument (file or db), and we store it in a bit-wise manner.
 */
const char *example_set_action(cmd_parms *cmd,
   void *cfg,
   const char *arg1,
   const char *arg2)
{
    if(!strcasecmp(arg1, "file")){
      config.typeOfAction = 0x01;
    } else {
      config.typeOfAction = 0x02;
    }
    if(!strcasecmp(arg2, "deny")){
      config.typeOfAction += 0x10;
    } else {
      config.typeOfAction += 0x20;
    }
    return NULL;
}

/*
* The directive structure for our name tag:
*==============================================================================*/
static const command_rec example_directives[] =
{
  AP_INIT_TAKE1("exampleEnabled", example_set_enabled, NULL, RSRC_CONF, "Enable or disable mod_example"),
  AP_INIT_TAKE1("examplePath", example_set_path, NULL, RSRC_CONF, "The path to whatever"),
  AP_INIT_TAKE2("exampleAction", example_set_action, NULL, RSRC_CONF, "Special action value!",
    NULL)
};

/*==============================================================================
Our module handler:
==============================================================================*/
static int example_handler(request_rec *r)
{
  if(!r->handler || strcmp(r->handler, "example-handler")){
    ap_set_content_type(r, "text/plain");
    ap_rprintf(r, "Enabled: %u\n", config.enabled);
    ap_rprintf(r, "Path: %s\n", config.path);
    ap_rprintf(r, "TypeOfAction: %x\n", config.typeOfAction);
    return OK;
  }
}
The hook registration function (also initializes the default config values):

```c
static void register_hooks(apr_pool_t *pool)
{
    config.enabled = 1;
    config.path = "/foo/bar";
    config.typeOfAction = 3;
    ap_hook_handler(example_handler, NULL, NULL, APR_HOOK_LAST);
}
```

Our module name tag:

```c
module AP_MODULE_DECLARE_DATA example_module =
{
    STANDARD20_MODULE_STUFF,
    NULL, /* Per-directory configuration handler */
    NULL, /* Merge handler for per-directory configurations */
    NULL, /* Per-server configuration handler */
    NULL, /* Merge handler for per-server configurations */
    example_directives, /* Any directives we may have for httpd */
    register_hooks /* Our hook registering function */
};
```

In our httpd.conf file, we can now change the hard-coded configuration by adding a few lines:

```
ExampleEnabled On
ExamplePath "/usr/bin/foo"
ExampleAction file allow
```
And thus we apply the configuration, visit /example on our website, and we see the configuration has adapted to what we wrote in our configuration file.
Introduction to context aware configurations

In Apache HTTP Server 2.4, different URLs, virtual hosts, directories etc can have very different meanings to the user of the server, and thus different contexts within which modules must operate. For example, let's assume you have this configuration set up for mod_rewrite:

```html
<Directory "/var/www">
    RewriteCond "%{HTTP_HOST}" "^example.com$"
    RewriteRule "(.*)" "http://www.example.com/$1"
</Directory>
<Directory "/var/www/sub">
    RewriteRule "^foobar$" "index.php?foobar=true"
</Directory>
```

In this example, you will have set up two different contexts for mod_rewrite:

1. Inside /var/www, all requests for http://example.com must go to http://www.example.com
2. Inside /var/www/sub, all requests for foobar must go to index.php?foobar=true

If mod_rewrite (or the entire server for that matter) wasn't context aware, then these rewrite rules would just apply to every and any request made, regardless of where and how they were made, but since the module can pull the context specific configuration straight from the server, it does not need to know itself, which of the directives are valid in this context, since the server takes care of this.

So how does a module get the specific configuration for the server,
directory or location in question? It does so by making one simple call:

```c
example_config *config = (example_config*) ap_get_module_config
```

That's it! Of course, a whole lot goes on behind the scenes, which we will discuss in this chapter, starting with how the server came to know what our configuration looks like, and how it came to be set up as it is in the specific context.

**Our basic configuration setup**

In this chapter, we will be working with a slightly modified version of our previous context structure. We will set a context variable that we can use to track which context configuration is being used by the server in various places:

```c
typedef struct {
    char    context[256];
    char    path[256];
    int     typeOfAction;
    int     enabled;
} example_config;
```

Our handler for requests will also be modified, yet still very simple:

```c
static int example_handler(request_rec *r)
{
    if(!r->handler || strcmp(r->handler, "example-handler")) return(DECLINED);
    example_config *config = (example_config*) ap_get_module_config(r->per_dir_config, &example_module);
    ap_set_content_type(r, "text/plain");
    ap_rprintf("Enabled: %u\n", config->enabled);
    ap_rprintf("Path: %s\n", config->path);
    ap_rprintf("TypeOfAction: %x\n", config->typeOfAction);
}
Choosing a context

Before we can start making our module context aware, we must first define, which contexts we will accept. As we saw in the previous chapter, defining a directive required five elements be set:

```
AP_INIT_TAKE1("exampleEnabled", example_set_enabled, NULL, RSRC_CONF, "Enable or disable mod_example")
```

The RSRC_CONF definition told the server that we would only allow this directive in a global server context, but since we are now trying out a context aware version of our module, we should set this to something more lenient, namely the value ACCESS_CONF, which lets us use the directive inside `<Directory>` and `<Location>` blocks. For more control over the placement of your directives, you can combine the following restrictions together to form a specific rule:

- **RSRC_CONF**: Allow in .conf files (not .htaccess) outside `<Directory>` or `<Location>`
- **ACCESS_CONF**: Allow in .conf files (not .htaccess) inside `<Directory>` or `<Location>`
- **OR_OPTIONS**: Allow in .conf files and .htaccess when AllowOverride Options is set
- **OR_FILEINFO**: Allow in .conf files and .htaccess when AllowOverride FileInfo is set
- **OR_AUTHCFG**: Allow in .conf files and .htaccess when AllowOverride AuthConfig is set
OR_INDEXES: Allow in .conf files and .htaccess when AllowOverride Indexes is set
OR_ALL: Allow anywhere in .conf files and .htaccess

Using the server to allocate configuration slots

A much smarter way to manage your configurations is by letting the server help you create them. To do so, we must first start off by changing our name tag to let the server know, that it should assist us in creating and managing our configurations. Since we have chosen the per-directory (or per-location) context for our module configurations, we'll add a per-directory creator and merger function reference in our tag:

```
module AP_MODULE_DECLARE_DATA example_module {
    STANDARD20_MODULE_STUFF,
    create_dir_conf, /* Per-directory configuration handler */
    merge_dir_conf, /* Merge handler for per-directory configurations */
    NULL, /* Per-server configuration handler */
    NULL, /* Merge handler for per-server configurations */
directives, /* Any directives we may have for httpd */
    register_hooks /* Our hook registering function */
};
```

Creating new context configurations

Now that we have told the server to help us create and manage configurations, our first step is to make a function for creating new, blank configurations. We do so by creating the function we just referenced in our name tag as the Per-directory configuration handler:

```
void *create_dir_conf(apr_pool_t *pool, char *context) {
```

Merging configurations

Our next step in creating a context aware configuration is merging configurations. This part of the process particularly applies to scenarios where you have a parent configuration and a child, such as the following:

```html
<Directory "/var/www">
    ExampleEnabled On
    ExamplePath "/foo/bar"
    ExampleAction file allow
</Directory>
<Directory "/var/www/subdir">
    ExampleAction file deny
</Directory>
```

In this example, it is natural to assume that the directory `/var/www/subdir` should inherit the values set for the `/var/www` directory, as we did not specify an `ExampleEnabled` nor an `ExamplePath` for this directory. The server does not presume to know if this is true, but cleverly does the following:

1. Creates a new configuration for `/var/www`
2. Sets the configuration values according to the directives given for /var/www

3. Creates a new configuration for /var/www/subdir

4. Sets the configuration values according to the directives given for /var/www/subdir

5. **Proposes a merge** of the two configurations into a new configuration for /var/www/subdir

This proposal is handled by the `merge_dir_conf` function we referenced in our name tag. The purpose of this function is to assess the two configurations and decide how they are to be merged:

```c
void *merge_dir_conf(apr_pool_t *pool, void *BASE, void *ADD) {
    example_config *base = (example_config *)BASE; /* This is what was set in the parent context */
    example_config *add = (example_config *)ADD; /* This is what is set in the new context */
    example_config *conf = (example_config *)create_dir_conf(pool, "Merged configuration"); /* This will be the merged configuration */

    /* Merge configurations */
    conf->enabled = (add->enabled == 0) ? base->enabled : add->enabled;
    conf->typeOfAction = add->typeOfAction ? add->typeOfAction : base->typeOfAction;
    strcpy(conf->path, strlen(add->path) ? add->path : base->path);

    return conf;
}
```

**Trying out our new context aware configurations**

Now, let's try putting it all together to create a new module that is context aware. First off, we'll create a configuration that lets us test how the module works:

```html
<Location "/a"
```
Then we'll assemble our module code. Note, that since we are now using our name tag as reference when fetching configurations in our handler, I have added some prototypes to keep the compiler happy:

```c
/*$6
   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   *  mod_example_config.c
   ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
   */

#include <stdio.h>
#include "apr_hash.h"
#include "ap_config.h"
#include "ap_provider.h"
#include "httpd.h"
#include "http_core.h"
#include "http_config.h"
```
#include "http_log.h"
#include "http_protocol.h"
#include "http_request.h"

/*
 */

typedef struct
{
    char context[256];
    char path[256];
    int typeOfAction;
    int enabled;
}
example_config;

/*
 */

static int example_handler(request_rec *)
const char *example_set_enabled(cmd_parms *cmd,
const char *example_set_path(cmd_parms *
const char *example_set_action(cmd_parms *
void *create_dir_conf(apr_pool_t *p)
void *merge_dir_conf(apr_pool_t *p)
static void *register_hooks(apr_pool_t *p)

/*
 */
static const command_rec directives[] = {
    AP_INIT_TAKE1("exampleEnabled", example_set_enabled, NULL, ACCESS_CONF, "Enable or disable mod_example"),
    AP_INIT_TAKE1("examplePath", example_set_path, NULL, ACCESS_CONF, "The path to whatever"),
    AP_INIT_TAKE2("exampleAction", example_set_action, NULL, ACCESS_CONF, "Special action value!")
};

/*$1
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Our name tag
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
module AP_MODULE_DECLARE_DATA example_module = {
    STANDARD20_MODULE_STUFF,
    create_dir_conf, /* Per-directory configuration handler */
    merge_dir_conf, /* Merge handler for per-directory configurations */
    NULL, /* Per-server configuration handler */
    NULL, /* Merge handler for per-server configurations */
    directives, /* Any directives we may have for httpd */
    register_hooks /* Our hook registering function */
};

/*
=======================================================================================================================
Hook registration function
=======================================================================================================================*/
static void register_hooks(apr_pool_t *pool) {
    ap_hook_handler(example_handler, NULL, NULL, APR_HOOK_LAST);
}
/*
Our example web service handler

static int example_handler(request_rec *r)
{
    if(!r->handler || strcmp(r->handler, "example-handler"))
        return(DECLINED);

    example_config *config = (example_config*)ap_get_module_config(r->per_dir_config, &example_module);

    ap_set_content_type(r, "text/plain");
    ap_rprintf(r, "Enabled: %u\n", config->enabled);
    ap_rprintf(r, "Path: %s\n", config->path);
    ap_rprintf(r, "TypeOfAction: %x\n", config->typeOfAction);
    ap_rprintf(r, "Context: %s\n", config->context);
    return OK;
}

Handler for the "exampleEnabled" directive

const char *example_set_enabled(cmd_parms *cmd, void *cfg, const char *arg)
{
    example_config *conf = (example_config*)cfg;

    if(conf)
    {
        if(!strcasecmp(arg, "on"))
            conf->enabled = 1;
        else
            conf->enabled = 0;
    }

    return conf;
}
Handler for the "examplePath" directive

const char *example_set_path(cmd_parms *cmd, void *cfg, const char *arg)
{
    example_config *conf = (example_config *) cfg;
    if(conf)
    {
        strcpy(conf->path, arg);
    }
    return NULL;
}

Handler for the "exampleAction" directive;
Let's pretend this one takes one argument (file or db), and a second (deny or allow), and we store it in a bit-wise manner.

const char *example_set_action(cmd_parms *cmd, void *cfg, const char *arg1, const char *arg2)
{
    example_config *conf = (example_config *) cfg;
    /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
    /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
    if(conf)
    {
        strcpy(conf->path, arg);
    }
    return NULL;
}
if(conf)
{
    {
        if(!strcasecmp(arg1, "file"))
            conf->typeOfAction = 0x01;
        else
            conf->typeOfAction = 0x02;
        if(!strcasecmp(arg2, "deny"))
            conf->typeOfAction += 0x10;
        else
            conf->typeOfAction += 0x20;
    }
}

return NULL;

/*

=======================================================================================================================
Function for creating new configurations for per-directory contexts
=======================================================================================================================*/

void *create_dir_conf(apr_pool_t *pool, char *context)
{
    context = context ? context : "Newly created configuration";
    /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
    example_config *cfg = apr_palloc(pool, sizeof(example_config));
    /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
    if(cfg)
    {
        /* Set some default values */
        strcpy(cfg->context, context);
        cfg->enabled = 0;
        memset(cfg->path, 0, 256);
        /*~~~~~~~~~*/
    }
    /*~~~~~~~~~*/
}
cfg->typeOfAction = 0x00;
}
}
return cfg;

/*
=======================================================================================================================
Merging function for configurations
=======================================================================================================================
*/
void *merge_dir_conf(apr_pool_t *pool, void *BASE, void *ADD)
{
    /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
    example_config *base = (example_config*) BASE;
    example_config *add = (example_config*) ADD;
    example_config *conf = (example_config*) create_dir_conf(pool, "Merged configuration");
    /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
    conf->enabled = (add->enabled == 0) ? base->enabled : add->enabled;
    conf->typeOfAction = add->typeOfAction ? add->typeOfAction : base->typeOfAction;
    strcpy(conf->path, strlen(add->path) ? add->path : base->path);
    return conf;
}
We have now looked at how to create simple modules for Apache HTTP Server 2.4 and configuring them. What you do next is entirely up to you, but it is my hope that something valuable has come out of reading this documentation. If you have questions on how to further develop modules, you are welcome to join our mailing lists or check out the rest of our documentation for further tips.
Retrieve variables from POST form data

typedef struct {
    const char *key;
    const char *value;
} keyValuePair;

keyValuePair *readPost(request_rec *r) {
    apr_array_header_t *pairs = NULL;
    apr_off_t len;
    apr_size_t size;
    int res;
    int i = 0;
    char *buffer;
    keyValuePair *kvp;

    res = ap_parse_form_data(r, NULL, &pairs);
    if (res != OK || !pairs) return NULL; /* Return NULL if we failed or if there are is no POST data */
    kvp = apr_pcalloc(r->pool, sizeof(keyValuePair) * (pairs->nelts + 1));
    while (pairs && !apr_is_empty_array(pairs)) {
        ap_form_pair_t *pair = (ap_form_pair_t *)apr_array_pop(pairs);
        apr_brigade_length(pair->value, 1, &len);
        size = (apr_size_t)len;
        buffer = apr_palloc(r->pool, size + 1);
        apr_brigade_flatten(pair->value, buffer, &size);
        buffer[len] = 0;
        kvp[i].key = apr_pstrdup(r->pool, pair->name);
        kvp[i].value = buffer;
        i++;
    }
    return kvp;
}

static int example_handler(request_rec *r) {
}
Printing out every HTTP header received

```c
static int example_handler(request_rec *r) {
    /*~~~~~~~~~~~~~~~~~~~~~~~~*/
    const apr_array_header_t *fields;
    int i;
    apr_table_entry_t *e = 0;
    /*~~~~~~~~~~~~~~~~~~~~~~~~*/
    
    fields = apr_table_elts(r->headers_in);
    e = (apr_table_entry_t *) fields->elts;
    for (i = 0; i < fields->nelts; i++) {
        /*~~~~~~~~~~~~~~~~~~~~~~~~*/
        
        } 
    return OK;
    /*~~~~~~~~~~~~~~~~~~~~~~~~*/
```
Reading the request body into memory

```c
static int util_read(request_rec *r, const char **rbuf, apr_off_t *size)
{
    /*~~~~~~~~*/
    int rc = OK;
    /*~~~~~~~~*/

    if((rc = ap_setup_client_block(r, REQUEST_CHUNKED_ERROR)))
    {
        return(rc);
    }

    if(ap_should_client_block(r)) {
        /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
        char argsbuffer[HUGE_STRING_LEN];
        apr_off_t rsize, len_read, rpos = 0;
        apr_off_t length = r->remaining;
        /*~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/

        *rbuf = (const char *) apr_pcalloc(r->pool, *size = length;
        while((len_read = ap_get_client_block(r, argsbuffer, sizeof(argsbuffer))) > 0)
        {
            if((rpos + len_read) > length)
            {
                rsize = length - rpos;
            }
            else {
                rsize = len_read;
            }
```
memcpy((char *)rbuf + rpos, argsbuffer, (size_t)rsize);

rpos += rsize;
}
}
return(rc);
}

static int example_handler(request_rec *r)
{
    /*~~~~~~~~~~~~~~~~*/
    apr_off_t size;
    const char *buffer;
    /*~~~~~~~~~~~~~~~~*/

    if(util_read(r, &buffer, &size) == OK)
        ap_rprintf(r, "We read a request body that was %" APR_OFF_T_FMT " bytes long", size);
    }
    return OK;
}
Frequently Asked Questions

The FAQ has been moved to the HTTP Server Wiki.

Copyright 2017 The Apache Software Foundation.
Licensed under the Apache License, Version 2.0.
Redirecting and Remapping with mod_rewrite

This document supplements the `mod_rewrite` reference documentation. It describes how you can use `mod_rewrite` to redirect and remap request. This includes many examples of common uses of mod_rewrite, including detailed descriptions of how each works.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

See also

- Module documentation
- mod_rewrite introduction
- Controlling access
- Virtual hosts
- Proxying
- Using RewriteMap
- Advanced techniques
- When not to use mod_rewrite
Description:
Assume we have recently renamed the page foo.html to bar.html and now want to provide the old URL for backward compatibility. However, we want that users of the old URL even not recognize that the pages was renamed - that is, we don't want the address to change in their browser.

Solution:
We rewrite the old URL to the new one internally via the following rule:

```
RewriteEngine on
RewriteRule "^/foo\.html$" "/bar.html" [PT]
```
Description:
Assume again that we have recently renamed the page foo.html to bar.html and now want to provide the old URL for backward compatibility. But this time we want that the users of the old URL get hinted to the new one, i.e. their browsers Location field should change, too.

Solution:
We force a HTTP redirect to the new URL which leads to a change of the browsers and thus the users view:

```
RewriteEngine on
RewriteRule '^/foo\.html$' "bar.html"
```

Discussion
In this example, as contrasted to the internal example above, we can simply use the Redirect directive. mod_rewrite was used in that earlier example in order to hide the redirect from the client:

```
Redirect "/foo.html" "bar.html"
```
Description:
If a resource has moved to another server, you may wish to have URLs continue to work for a time on the old server while people update their bookmarks.

Solution:
You can use `mod_rewrite` to redirect these URLs to the new server, but you might also consider using the Redirect or RedirectMatch directive.

```
#With mod_rewrite
RewriteEngine on
RewriteRule "^/docs/(.+)
" http://new.example.com/docs/$1 [R,L]

#With RedirectMatch
RedirectMatch "^/docs/(.*)" http://new.example.com/docs/

#With Redirect
Redirect "/docs/" http://new.example.com/docs/
```
Description:
How can we transform a static page foo.html into a dynamic variant foo.cgi in a seamless way, i.e. without notice by the browser/user.

Solution:
We just rewrite the URL to the CGI-script and force the handler to be cgiscript so that it is executed as a CGI program. This way a request to /~quux/foo.html internally leads to the invocation of /~quux/foo.cgi.

```
RewriteEngine on
RewriteBase "/~quux/"
RewriteRule "^foo\.html$" "foo.cgi"
```
**Description:**
How can we make URLs backward compatible (still existing virtually) after migrating document.YYYY to document.XXXX, e.g. after translating a bunch of .html files to .php?

**Solution:**
We rewrite the name to its basename and test for existence of the new extension. If it exists, we take that name, else we rewrite the URL to its original state.

```plaintext
#    backward compatibility ruleset for
#    rewriting document.html to document.
#    when and only when document.php exists
<Directory "/var/www/htdocs">
    RewriteEngine on
    RewriteBase "/var/www/htdocs"

    RewriteCond "$1.php"  -f
    RewriteCond "$1.html"  !-f
    RewriteRule "^(.*).html$" "$1.php"
</Directory>
```

**Discussion**
This example uses an often-overlooked feature of mod_rewrite, by taking advantage of the order of execution of the ruleset. In particular, mod_rewrite evaluates the left-hand-side of the RewriteRule before it evaluates the RewriteCond directives. Consequently, $1 is already defined by the time the RewriteCond directives are evaluated. This allows us to test for the existence of the original (document.html) and target (document.php) files using the same base filename.
This ruleset is designed to use in a per-directory context (In a <Directory> block or in a .htaccess file), so that the -f checks are looking at the correct directory path. You may need to set a RewriteBase directive to specify the directory base that you're working in.
Description:
The goal of this rule is to force the use of a particular hostname, in preference to other hostnames which may be used to reach the same site. For example, if you wish to force the use of www.example.com instead of example.com, you might use a variant of the following recipe.

Solution:
The very best way to solve this doesn't involve mod_rewrite at all, but rather uses the Redirect directive placed in a virtual host for the non-canonical hostname(s).

```xml
<VirtualHost *:80>
  ServerName undesired.example.com
  ServerAlias example.com notthis.example.com
  Redirect "/" "http://www.example.com/
</VirtualHost>

<VirtualHost *:80>
  ServerName www.example.com
</VirtualHost>
```

You can alternatively accomplish this using the `<If>` directive:

```xml
<If "${HTTP_HOST} != 'www.example.com'">
  Redirect "/" "http://www.example.com"
</If>
```

Or, for example, to redirect a portion of your site to HTTPS, you might do the following:
If, for whatever reason, you still want to use mod_rewrite - if, for example, you need this to work with a larger set of RewriteRules - you might use one of the recipes below.

For sites running on a port other than 80:

```apache
RewriteCond %{HTTP_HOST} !^www\.(example.com)
RewriteCond %{HTTP_HOST} !^$
RewriteCond %{SERVER_PORT} !^80$
RewriteRule ^/(.*) http://www.example.com:%{SERVER_PORT}/$1 [L,R,NE]
```

And for a site running on port 80

```apache
RewriteCond %{HTTP_HOST} !^www\.(example.com)
RewriteCond %{HTTP_HOST} !^$
RewriteRule ^/(.*) http://www.example.com/$1 [L,R,NE]
```

If you wanted to do this generically for all domain names - that is, if you want to redirect `example.com` to `www.example.com` for all possible values of `example.com`, you could use the following recipe:

```apache
RewriteCond %{HTTP_HOST} !^www\.$ [NC]
RewriteCond %{HTTP_HOST} !^$
RewriteRule ^/(.*) http://www.%{HTTP_HOST}/$1 [L,R,NE]
```
These rulesets will work either in your main server configuration file, or in a .htaccess file placed in the DocumentRoot of the server.
**Description:**
A particular resource might exist in one of several places, and we want to look in those places for the resource when it is requested. Perhaps we've recently rearranged our directory structure, dividing content into several locations.

**Solution:**
The following ruleset searches in two directories to find the resource, and, if not finding it in either place, will attempt to just serve it out of the location requested.

```perl
RewriteEngine on

# first try to find it in dir1/...
# ...and if found stop and be happy:
RewriteCond %{DOCUMENT_ROOT}/dir1/ %{REQUEST_URI} -f
RewriteRule "^(.+)

# second try to find it in dir2/...
# ...and if found stop and be happy:
RewriteCond %{DOCUMENT_ROOT}/dir2/ %{REQUEST_URI} -f
RewriteRule "^(.+)

# else go on for other Alias or Script
# etc.
RewriteRule "^" "-" [PT]
```
Description:
We have numerous mirrors of our website, and want to redirect people to the one that is located in the country where they are located.

Solution:
Looking at the hostname of the requesting client, we determine which country they are coming from. If we can't do a lookup on their IP address, we fall back to a default server.

We'll use a RewriteMap directive to build a list of servers that we wish to use.

```
HostnameLookups on
RewriteEngine on
RewriteMap multiplex "txt:/path/to/map.mirrors"
RewriteCond " %{REMOTE_HOST}" "([a-z]+)$" [NC]
RewriteRule " ^/(.*)$" "${multiplex:%1}|http://www.example.com/}$1" [R,L]
```

```#
## map.mirrors -- Multiplexing Map

de http://www.example.de/
uk http://www.example.uk/
com http://www.example.com/
##EOF##
```

Discussion
This ruleset relies on HostNameLookups being set on, which can be a significant performance hit.

The RewriteCond directive captures the last portion of the hostname of the requesting client - the country code - and the
following RewriteRule uses that value to look up the appropriate mirror host in the map file.
Description:
We wish to provide different content based on the browser, or user-agent, which is requesting the content.

Solution:
We have to decide, based on the HTTP header "User-Agent", which content to serve. The following config does the following: If the HTTP header "User-Agent" contains "Mozilla/3", the page foo.html is rewritten to foo.NS.html and the rewriting stops. If the browser is "Lynx" or "Mozilla" of version 1 or 2, the URL becomes foo.20.html. All other browsers receive page foo.32.html. This is done with the following ruleset:

```plaintext
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/3
RewriteRule ^foo\.html$ foo.NS.html

RewriteCond %{HTTP_USER_AGENT} ^Lynx/
[OR]
RewriteCond %{HTTP_USER_AGENT} ^Mozilla/12
RewriteRule ^foo\.html$ foo.20.html

RewriteRule ^foo\.html$ foo.32.html
```

...
Description:
On some webservers there is more than one URL for a resource. Usually there are canonical URLs (which are actually used and distributed) and those which are just shortcuts, internal ones, and so on. Independent of which URL the user supplied with the request, they should finally see the canonical one in their browser address bar.

Solution:
We do an external HTTP redirect for all non-canonical URLs to fix them in the location view of the Browser and for all subsequent requests. In the example ruleset below we replace /puppies and /canines by the canonical /dogs.

```
RewriteRule "^/(puppies|canines)/(.*)" /dogs/$2 [R]
```

Discussion:
This should really be accomplished with Redirect or RedirectMatch directives:

```
RedirectMatch "^/(puppies|canines)/(.*)"
```
Description:

Usually the DocumentRoot of the webserver directly relates to the URL "/". But often this data is not really of top-level priority. For example, you may wish for visitors, on first entering a site, to go to a particular subdirectory /about/. This may be accomplished using the following ruleset:

Solution:

We redirect the URL / to /about/:

```
RewriteEngine on
RewriteRule "^/$" "about/" [R]
```

Note that this can also be handled using the RedirectMatch directive:

```
RedirectMatch "^/$" "http://example.com/about/
```

Note also that the example rewrites only the root URL. That is, it rewrites a request for http://example.com/, but not a request for http://example.com/page.html. If you have in fact changed your document root - that is, if all of your content is in fact in that subdirectory, it is greatly preferable to simply change your DocumentRoot directive, or move all of the content up one directory, rather than rewriting URLs.
Description:

You want a single resource (say, a certain file, like index.php) to handle all requests that come to a particular directory, except those that should go to an existing resource such as an image, or a css file.

Solution:

As of version 2.2.16, you should use the `FallbackResource` directive for this:

```xml
<Directory "/var/www/my_blog">
    FallbackResource "index.php"
</Directory>
```

However, in earlier versions of Apache, or if your needs are more complicated than this, you can use a variation of the following rewrite set to accomplish the same thing:

```xml
<Directory "/var/www/my_blog">
    RewriteBase "/my_blog"

    RewriteCond "/var/www/my_blog/%{REQUEST_FILENAME}" !-f
    RewriteCond "/var/www/my_blog/%{REQUEST_FILENAME}" !-d
    RewriteRule "^" "index.php" [PT]
</Directory>
```

If, on the other hand, you wish to pass the requested URI as a query string argument to index.php, you can replace that `RewriteRule` with:

```xml
RewriteRule "(.*)" "index.php?$1" [PT,QS]"
Note that these rulesets can be used in a `.htaccess` file, as well as in a `<Directory>` block.
Description:
You want to capture a particular value from a query string and either replace it or incorporate it into another component of the URL.

Solutions:
Many of the solutions in this section will all use the same condition, which leaves the matched value in the %2 backreference. %1 is the beginning of the query string (up to the key of interest), and %3 is the remainder. This condition is a bit complex for flexibility and to avoid double '&&' in the substitutions.

- This solution removes the matching key and value:

```
# Remove mykey=???
RewriteCond %{QUERY_STRING} "(.*(?:^|&))mykey=(\[^&\]*)&?(.*)&?$"
RewriteRule "(.*)" "$1%1%3"
```

- This solution uses the captured value in the URL substitution, discarding the rest of the original query by appending a '?':

```
# Copy from query string to PATH_INFO
RewriteCond %{QUERY_STRING} "(.*(?:^|&))mykey=(\[^&\]*)&?(.*)&?$"
RewriteRule "(.*)" "$1/products/%2/?"
```

- This solution checks the captured value in a subsequent condition:

```
# Capture the value of mykey in the condition
RewriteCond %{QUERY_STRING} "(.*(?:^|&))mykey=(\[^&\]*)&?(.*)&?$"
```
This solution shows the reverse of the previous ones, copying path components (perhaps PATH_INFO) from the URL into the query string.

```apache
copy
RewriteRule "^/?path/([^/]+)/([^/]+)" "/path?$1=$2" [PT]
```
Using mod_rewrite to control access

This document supplements the mod_rewrite reference documentation. It describes how you can use mod_rewrite to control access to various resources, and other related techniques. This includes many examples of common uses of mod_rewrite, including detailed descriptions of how each works.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

See also

Module documentation
mod_rewrite introduction
Redirection and remapping
Virtual hosts
Proxying
Using RewriteMap
Advanced techniques
When not to use mod_rewrite
Forbidding Image "Hotlinking"

Description:
The following technique forbids the practice of other sites including your images inline in their pages. This practice is often referred to as "hotlinking", and results in your bandwidth being used to serve content for someone else's site.

Solution:
This technique relies on the value of the HTTP_REFERER variable, which is optional. As such, it's possible for some people to circumvent this limitation. However, most users will experience the failed request, which should, over time, result in the image being removed from that other site.

There are several ways that you can handle this situation.

In this first example, we simply deny the request, if it didn't initiate from a page on our site. For the purpose of this example, we assume that our site is www.example.com.

```
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !www.example.com [NC]
RewriteRule \.(gif|jpg|png)$ - [F,NC]
```

In this second example, instead of failing the request, we display an alternate image instead.

```
RewriteCond %{HTTP_REFERER} !^$
RewriteCond %{HTTP_REFERER} !www.example.com [NC]
RewriteRule \.(gif|jpg|png)$ /images/go-away.png [R,NC]
```

In the third example, we redirect the request to an image on some other site.
Of these techniques, the last two tend to be the most effective in getting people to stop hotlinking your images, because they will simply not see the image that they expected to see.

**Discussion:**

If all you wish to do is deny access to the resource, rather than redirecting that request elsewhere, this can be accomplished without the use of mod_rewrite:

```apache
SetEnvIf Referer "example\.com" localreferer
<FilesMatch "\.(jpg|png|gif)$">
  Require env localreferer
</FilesMatch>
```
Description:
In this recipe, we discuss how to block persistent requests from a particular robot, or user agent.

The standard for robot exclusion defines a file, /robots.txt that specifies those portions of your website where you wish to exclude robots. However, some robots do not honor these files.

Note that there are methods of accomplishing this which do not use mod_rewrite. Note also that any technique that relies on the clients USER_AGENT string can be circumvented very easily, since that string can be changed.

Solution:
We use a ruleset that specifies the directory to be protected, and the client USER_AGENT that identifies the malicious or persistent robot.

In this example, we are blocking a robot called NameOfBadRobot from a location /secret/files. You may also specify an IP address range, if you are trying to block that user agent only from the particular source.

```
RewriteCond %{HTTP_USER_AGENT} "^NameOfBadRobot"
RewriteCond %{REMOTE_ADDR} "=123.45.67.8-9"
RewriteRule ^/secret/files/ "-" [F]
```

Discussion:
Rather than using mod_rewrite for this, you can accomplish the same end using alternate means, as illustrated here:
As noted above, this technique is trivial to circumvent, by simply modifying the USER_AGENT request header. If you are experiencing a sustained attack, you should consider blocking it at a higher level, such as at your firewall.
Description:
We wish to maintain a blacklist of hosts, rather like hosts.deny, and have those hosts blocked from accessing our server.

Solution:

```
RewriteEngine on
RewriteMap hosts-deny "txt:/path/to/hosts.deny"
RewriteCond %{hosts-deny:%{REMOTE_ADDR}|NOT-FOUND} !=NOT-FOUND [OR]
RewriteCond %{hosts-deny:%{REMOTE_HOST}|NOT-FOUND} !=NOT-FOUND
RewriteRule ^ - [F]
```

###
### hosts.deny
###
### ATTENTION! This is a map, not a list, even when we treat it as such.
### mod_rewrite parses it for key/value pairs, so at least a
### dummy value "-" must be present for each entry.
###
193.102.180.41 -
bsdtil.sdm.de -
192.162.40 -

Discussion:
The second RewriteCond assumes that you have HostNameLookups turned on, so that client IP addresses will be resolved. If that's not the case, you should drop the second RewriteCond, and drop the [OR] flag from the first RewriteCond.
Description:
Redirect requests based on the Referer from which the request came, with different targets per Referer.

Solution:
The following ruleset uses a map file to associate each Referer with a redirection target.

```
RewriteMap deflector "txt:/path/to/deflector.map"
RewriteCond %{HTTP_REFERER} !="" [NC]
RewriteCond ${deflector:%{HTTP_REFERER}} "=-" [NC]
RewriteRule ^%{HTTP_REFERER} [R,L]
RewriteCond %{HTTP_REFERER} !="" [NC]
RewriteCond ${deflector:%{HTTP_REFERER}|NOT-FOUND} "!=NOT-FOUND" [NC]
RewriteRule ^%{HTTP_REFERER} [R,L]
```

The map file lists redirection targets for each referer, or, if we just wish to redirect back to where they came from, a "-" is placed in the map:

```
##
## deflector.map
##
http://badguys.example.com/bad/index.html				-
http://badguys.example.com/bad/index2.html			-
http://badguys.example.com/bad/index3.html			http://somewhere.example.com/
```
Using mod_rewrite for Proxying

This document supplements the mod_rewrite reference documentation. It describes how to use the RewriteRule's [P] flag to proxy content to another server. A number of recipes are provided that describe common scenarios.

See also

Module documentation
mod_rewrite introduction
Redirection and remapping
Controlling access
Virtual hosts
Using RewriteMap
Advanced techniques
When not to use mod_rewrite
Proxying Content with mod_rewrite

**Description:**
mod_rewrite provides the [P] flag, which allows URLs to be passed, via mod_proxy, to another server. Two examples are given here. In one example, a URL is passed directly to another server, and served as though it were a local URL. In the other example, we proxy missing content to a back-end server.

**Solution:**
To simply map a URL to another server, we use the [P] flag, as follows:

```apache
RewriteEngine on
RewriteBase "/products/
RewriteRule "^widget/(.*)$" "http://product.example.com/widget/$1" [P]
ProxyPassReverse "/products/widget/" http://product.example.com/widget/
```

In the second example, we proxy the request only if we can't find the resource locally. This can be very useful when you're migrating from one server to another, and you're not sure if all the content has been migrated yet.

```apache
RewriteCond %{REQUEST_FILENAME} !-f
RewriteCond %{REQUEST_FILENAME} !-d
RewriteRule "^/(.*)" "http://old.example.com/$1" [P]
ProxyPassReverse "/" "http://old.example.com/
```

**Discussion:**
In each case, we add a `ProxyPassReverse` directive to ensure that any redirects issued by the backend are correctly passed on to the client.
Consider using either `ProxyPass` or `ProxyPassMatch` whenever possible in preference to `mod_rewrite`.
Advanced Techniques with mod_rewrite

This document supplements the mod_rewrite reference documentation. It provides a few advanced techniques using mod_rewrite.

Note that many of these examples won’t work unchanged in your particular server configuration, so it’s important that you understand them, rather than merely cutting and pasting the examples into your configuration.

See also

Module documentation
mod_rewrite introduction
Redirection and remapping
Controlling access
Virtual hosts
Proxying
Using RewriteMap
When not to use mod_rewrite
Description:
A common technique for distributing the burden of server load or storage space is called "sharding". When using this method, a front-end server will use the URL to consistently "shard" users or objects to separate backend servers.

Solution:
A mapping is maintained, from users to target servers, in external map files. They look like:

```
user1  physical_host_of_user1
user2  physical_host_of_user2
::
```

We put this into a `map.users-to-hosts` file. The aim is to map;

```
/u/user1/anypath
```

to

```
http://physical_host_of_user1/u/user/anypath
```

thus every URL path need not be valid on every backend physical host. The following ruleset does this for us with the help of the map files assuming that server0 is a default server which will be used if a user has no entry in the map:

```
RewriteEngine on
RewriteMap users-to-hosts "txt:/path/to/map.users-to-hosts"
RewriteRule "^/u/([^/]+)/?(.*)" "http://${users-to-hosts:$1|server0}/u/$1/$2"
```

See the `RewriteMap` documentation for more discussion of the
syntax of this directive.
Description:
We wish to dynamically generate content, but store it statically once it is generated. This rule will check for the existence of the static file, and if it's not there, generate it. The static files can be removed periodically, if desired (say, via cron) and will be regenerated on demand.

Solution:
This is done via the following ruleset:

```
# This example is valid in per-directory context only
RewriteCond %{REQUEST_URI} !-U
RewriteRule ^(.+)\.html$ /regenerate_page.cgi [PT,L]
```

The -U operator determines whether the test string (in this case, REQUEST_URI) is a valid URL. It does this via a subrequest. In the event that this subrequest fails - that is, the requested resource doesn't exist - this rule invokes the CGI program /regenerate_page.cgi, which generates the requested resource and saves it into the document directory, so that the next time it is requested, a static copy can be served.

In this way, documents that are infrequently updated can be served in static form. If documents need to be refreshed, they can be deleted from the document directory, and they will then be regenerated the next time they are requested.
Description:
We wish to randomly distribute load across several servers using mod_rewrite.

Solution:
We'll use RewriteMap and a list of servers to accomplish this.

```apache
RewriteEngine on
RewriteMap lb "rnd:/path/to/serverlist.txt"
RewriteRule "^/(.*)" "http://${lb:servers}/$1" [P,L]
```

serverlist.txt will contain a list of the servers:

```
## serverlist.txt
servers one.example.com|two.example.com|three.example.com
```

If you want one particular server to get more of the load than the others, add it more times to the list.

Discussion
Apache comes with a load-balancing module - mod_proxy_balancer - which is far more flexible and featureful than anything you can cobble together using mod_rewrite.
Description:
Some sites with thousands of users use a structured homedir layout, *i.e.* each homedir is in a subdirectory which begins (for instance) with the first character of the username. So, 

/~larry/anypath is 
/home/l/larry/public_html/anypath while 
/~waldo/anypath is 
/home/w/waldo/public_html/anypath.

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

```
RewriteEngine on
RewriteRule "^/~([a-z][a-z0-9]+)(.*)" /home/
```
Description:
By default, redirecting to an HTML anchor doesn't work, because mod_rewrite escapes the # character, turning it into %23. This, in turn, breaks the redirection.

Solution:
Use the [NE] flag on the RewriteRule. NE stands for No Escape.

Discussion:
This technique will of course also work with other special characters that mod_rewrite, by default, URL-encodes.
Description:
We wish to use mod_rewrite to serve different content based on the time of day.

Solution:
There are a lot of variables named TIME_xxx for rewrite conditions. In conjunction with the special lexicographic comparison patterns "<STRING,"STRING" and "=STRING" we can do time-dependent redirects:

```
RewriteEngine on
RewriteCond %{TIME_HOUR}%{TIME_MIN} ">0700"
RewriteCond %{TIME_HOUR}%{TIME_MIN} "<1900"
RewriteRule ^foo\.html$ foo.day.html [L]
RewriteRule ^foo\.html$ foo.night.html
```

This provides the content of foo.day.html under the URL foo.html from 07:01-18:59 and at the remaining time the contents of foo.night.html.

**mod_cache**, intermediate proxies and browsers may each cache responses and cause the either page to be shown outside of the time-window configured. **mod_expires** may be used to control this effect. You are, of course, much better off simply serving the content dynamically, and customizing it based on the time of day.
Description:
At time, we want to maintain some kind of status when we perform a rewrite. For example, you want to make a note that you've done that rewrite, so that you can check later to see if a request can via that rewrite. One way to do this is by setting an environment variable.

Solution:
Use the [E] flag to set an environment variable.

```apache
cat
RewriteEngine on
RewriteRule "^/horse/(.*)" "/pony/$1"
```

Later in your ruleset you might check for this environment variable using a RewriteCond:

```
RewriteCond %{ENV:rewritten}" "=1"
```

Note that environment variables do not survive an external redirect. You might consider using the [CO] flag to set a cookie.
When not to use mod_rewrite

This document supplements the mod_rewrite reference documentation. It describes perhaps one of the most important concepts about mod_rewrite - namely, when to avoid using it.

mod_rewrite should be considered a last resort, when other alternatives are found wanting. Using it when there are simpler alternatives leads to configurations which are confusing, fragile, and hard to maintain. Understanding what other alternatives are available is a very important step towards mod_rewrite mastery.

Note that many of these examples won't work unchanged in your particular server configuration, so it's important that you understand them, rather than merely cutting and pasting the examples into your configuration.

The most common situation in which mod_rewrite is the right tool is when the very best solution requires access to the server configuration files, and you don't have that access. Some configuration directives are only available in the server configuration file. So if you are in a hosting situation where you only have .htaccess files to work with, you may need to resort to mod_rewrite.

See also

Module documentation
mod_rewrite introduction
Redirection and remapping
Controlling access
Virtual hosts
Proxying
Using RewriteMap
Advanced techniques
**mod_alias** provides the **Redirect** and **RedirectMatch** directives, which provide a means to redirect one URL to another. This kind of simple redirection of one URL, or a class of URLs, to somewhere else, should be accomplished using these directives rather than **RewriteRule**. **RedirectMatch** allows you to include a regular expression in your redirection criteria, providing many of the benefits of using **RewriteRule**.

A common use for **RewriteRule** is to redirect an entire class of URLs. For example, all URLs in the /one directory must be redirected to http://one.example.com/, or perhaps all http requests must be redirected to https.

These situations are better handled by the **Redirect** directive. Remember that **Redirect** preserves path information. That is to say, a redirect for a URL /one will also redirect all URLs under that, such as /one/two.html and /one/three/four.html.

To redirect URLs under /one to http://one.example.com, do the following:

```
Redirect "/one/" "http://one.example.com/"
```

To redirect one hostname to another, for example example.com to www.example.com, see the **Canonical Hostnames** recipe.

To redirect http URLs to https, do the following:

```
<VirtualHost *:80>
  ServerName www.example.com
  Redirect "/" "https://www.example.com/"
</VirtualHost>
```
The use of RewriteRule to perform this task may be appropriate if there are other RewriteRule directives in the same scope. This is because, when there are Redirect and RewriteRule directives in the same scope, the RewriteRule directives will run first, regardless of the order of appearance in the configuration file.

In the case of the http-to-https redirection, the use of RewriteRule would be appropriate if you don't have access to the main server configuration file, and are obliged to perform this task in a .htaccess file instead.
The **Alias** directive provides mapping from a URI to a directory - usually a directory outside of your **DocumentRoot**. Although it is possible to perform this mapping with **mod_rewrite**, **Alias** is the preferred method, for reasons of simplicity and performance.

**Using Alias**

Alias "/cats" "/var/www/virtualhosts/felines/htdocs"

The use of **mod_rewrite** to perform this mapping may be appropriate when you do not have access to the server configuration files. **Alias** may only be used in server or virtualhost context, and not in a `.htaccess` file.

Symbolic links would be another way to accomplish the same thing, if you have **Options FollowSymLinks** enabled on your server.
Although it is possible to handle virtual hosts with mod_rewrite, it is seldom the right way. Creating individual `<VirtualHost>` blocks is almost always the right way to go. In the event that you have an enormous number of virtual hosts, consider using `mod_vhost_alias` to create these hosts automatically.

Modules such as `mod_macro` are also useful for creating a large number of virtual hosts dynamically.

Using `mod_rewrite` for virtualhost creation may be appropriate if you are using a hosting service that does not provide you access to the server configuration files, and you are therefore restricted to configuration using `.htaccess` files.

See the virtual hosts with mod_rewrite document for more details on how you might accomplish this if it still seems like the right approach.
**RewriteRule** provides the `[P]` flag to pass rewritten URIs through **mod_proxy**.

```
RewriteRule "^/?images(.*\)" "http://imageserver.local/images$1"
```

However, in many cases, when there is no actual pattern matching needed, as in the example shown above, the **ProxyPass** directive is a better choice. The example here could be rendered as:

```
ProxyPass "/images/" "http://imageserver.local/images/
```

Note that whether you use **RewriteRule** or **ProxyPass**, you'll still need to use the **ProxyPassReverse** directive to catch redirects issued from the back-end server:

```
ProxyPassReverse "/images/" "http://imageserver.local/images/
```

You may need to use **RewriteRule** instead when there are other **RewriteRules** in effect in the same scope, as a **RewriteRule** will usually take effect before a **ProxyPass**, and so may preempt what you're trying to accomplish.
mod_rewrite is frequently used to take a particular action based on the presence or absence of a particular environment variable or request header. This can be done more efficiently using the `<If>`.

Consider, for example, the common scenario where RewriteRule is used to enforce a canonical hostname, such as www.example.com instead of example.com. This can be done using the `<If>` directive, as shown here:

```
<If "req('Host') != 'www.example.com'">
  Redirect "/" "http://www.example.com/"
</If>
```

This technique can be used to take actions based on any request header, response header, or environment variable, replacing mod_rewrite in many common scenarios.

See especially the expression evaluation documentation for a overview of what types of expressions you can use in `<If>` sections, and in certain other directives.
Apache mod_rewrite Technical Details

This document discusses some of the technical details of mod_rewrite and URL matching.

See also

- Module documentation
- mod_rewrite introduction
- Redirection and remapping
- Controlling access
- Virtual hosts
- Proxying
- Using RewriteMap
- Advanced techniques
- When not to use mod_rewrite
The Apache HTTP Server handles requests in several phases. At each of these phases, one or more modules may be called upon to handle that portion of the request lifecycle. Phases include things like URL-to-filename translation, authentication, authorization, content, and logging. (This is not an exhaustive list.)

mod_rewrite acts in two of these phases (or "hooks", as they are often called) to influence how URLs may be rewritten.

First, it uses the URL-to-filename translation hook, which occurs after the HTTP request has been read, but before any authorization starts. Secondly, it uses the Fixup hook, which is after the authorization phases, and after per-directory configuration files (.htaccess files) have been read, but before the content handler is called.

So, after a request comes in and a corresponding server or virtual host has been determined, the rewriting engine starts processing any mod_rewrite directives appearing in the per-server configuration. (i.e., in the main server configuration file and <Virtualhost> sections.) This happens in the URL-to-filename phase.

A few steps later, once the final data directories have been found, the per-directory configuration directives (.htaccess files and <Directory> blocks) are applied. This happens in the Fixup phase.

In each of these cases, mod_rewrite rewrites the REQUEST_URI either to a new URL, or to a filename.

In per-directory context (i.e., within .htaccess files and Directory blocks), these rules are being applied after a URL has already been translated to a filename. Because of this, the URL-
path that mod_rewrite initially compares RewriteRule directives against is the full filesystem path to the translated filename with the current directories path (including a trailing slash) removed from the front.

To illustrate: If rules are in /var/www/foo/.htaccess and a request for /foo/bar/baz is being processed, an expression like ^bar/baz$ would match.

If a substitution is made in per-directory context, a new internal subrequest is issued with the new URL, which restarts processing of the request phases. If the substitution is a relative path, the RewriteBase directive determines the URL-path prefix prepended to the substitution. In per-directory context, care must be taken to create rules which will eventually (in some future "round" of per-directory rewrite processing) not perform a substitution to avoid looping. (See RewriteLooping for further discussion of this problem.)

Because of this further manipulation of the URL in per-directory context, you'll need to take care to craft your rewrite rules differently in that context. In particular, remember that the leading directory path will be stripped off of the URL that your rewrite rules will see. Consider the examples below for further clarification.

<table>
<thead>
<tr>
<th>Location of rule</th>
<th>Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>VirtualHost section</td>
<td>RewriteRule &quot;^/images/(.+).jpg&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;/images/$1.gif&quot;</td>
</tr>
<tr>
<td>.htaccess file in document root</td>
<td>RewriteRule &quot;^images/(.+).jpg&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;images/$1.gif&quot;</td>
</tr>
<tr>
<td>.htaccess file in images directory</td>
<td>RewriteRule &quot;^(.+).jpg&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;$1.gif&quot;</td>
</tr>
</tbody>
</table>

For even more insight into how mod_rewrite manipulates URLs in
different contexts, you should consult the log entries made during rewriting.
Now when mod_rewrite is triggered in these two API phases, it reads the configured rulesets from its configuration structure (which itself was either created on startup for per-server context or during the directory walk of the Apache kernel for per-directory context). Then the URL rewriting engine is started with the contained ruleset (one or more rules together with their conditions). The operation of the URL rewriting engine itself is exactly the same for both configuration contexts. Only the final result processing is different.

The order of rules in the ruleset is important because the rewriting engine processes them in a special (and not very obvious) order. The rule is this: The rewriting engine loops through the ruleset rule by rule (RewriteRule directives) and when a particular rule matches it optionally loops through existing corresponding conditions (RewriteCond directives). For historical reasons the conditions are given first, and so the control flow is a little bit long-winded. See Figure 1 for more details.
First the URL is matched against the Pattern of each rule. If it fails, mod_rewrite immediately stops processing this rule, and continues with the next rule. If the Pattern matches, mod_rewrite looks for corresponding rule conditions (RewriteCond directives, appearing immediately above the RewriteRule in the configuration). If none are present, it substitutes the URL with a new value, which is constructed from the string Substitution, and goes on with its rule-looping. But if conditions exist, it starts an inner loop for processing them in the order that they are listed. For conditions, the logic is different: we don't match a pattern against the current URL. Instead we first create a string TestString by expanding variables, back-references, map lookups, etc. and then we try to match CondPattern against it. If the pattern doesn't match, the complete set of conditions and the corresponding rule fails. If the pattern matches, then the next condition is processed until no
more conditions are available. If all conditions match, processing is continued with the substitution of the URL with Substitution.
Guide to writing output filters

There are a number of common pitfalls encountered when writing output filters; this page aims to document best practice for authors of new or existing filters.

This document is applicable to both version 2.0 and version 2.2 of the Apache HTTP Server; it specifically targets RESOURCE-level or CONTENT_SET-level filters though some advice is generic to all types of filter.
Each time a filter is invoked, it is passed a bucket brigade, containing a sequence of buckets which represent both data content and metadata. Every bucket has a bucket type; a number of bucket types are defined and used by the httpd core modules (and the apr-util library which provides the bucket brigade interface), but modules are free to define their own types.

Output filters must be prepared to process buckets of non-standard types; with a few exceptions, a filter need not care about the types of buckets being filtered.

A filter can tell whether a bucket represents either data or metadata using the APR_BUCKET_IS_METADATA macro. Generally, all metadata buckets should be passed down the filter chain by an output filter. Filters may transform, delete, and insert data buckets as appropriate.

There are two metadata bucket types which all filters must pay attention to: the EOS bucket type, and the FLUSH bucket type. An EOS bucket indicates that the end of the response has been reached and no further buckets need be processed. A FLUSH bucket indicates that the filter should flush any buffered buckets (if applicable) down the filter chain immediately.

FLUSH buckets are sent when the content generator (or an upstream filter) knows that there may be a delay before more content can be sent. By passing FLUSH buckets down the filter chain immediately, filters ensure that the client is not kept waiting for pending data longer than necessary.

Filters can create FLUSH buckets and pass these down the filter chain if desired. Generating FLUSH buckets unnecessarily, or too frequently, can harm network utilisation since it may force large
numbers of small packets to be sent, rather than a small number of larger packets. The section on Non-blocking bucket reads covers a case where filters are encouraged to generate FLUSH buckets.

**Example bucket brigade**

HEAP FLUSH FILE EOS

This shows a bucket brigade which may be passed to a filter; it contains two metadata buckets (FLUSH and EOS), and two data buckets (HEAP and FILE).
For any given request, an output filter might be invoked only once and be given a single brigade representing the entire response. It is also possible that the number of times a filter is invoked for a single response is proportional to the size of the content being filtered, with the filter being passed a brigade containing a single bucket each time. Filters must operate correctly in either case.

An output filter which allocates long-lived memory every time it is invoked may consume memory proportional to response size. Output filters which need to allocate memory should do so once per response; see Maintaining state below.

An output filter can distinguish the final invocation for a given response by the presence of an EOS bucket in the brigade. Any buckets in the brigade after an EOS should be ignored.

An output filter should never pass an empty brigade down the filter chain. To be defensive, filters should be prepared to accept an empty brigade, and should return success without passing this brigade on down the filter chain. The handling of an empty brigade should have no side effects (such as changing any state private to the filter).

### How to handle an empty brigade

```c
apr_status_t dummy_filter(ap_filter_t *f, apr_bucket_brigade *b) {
    if (APR_BRIGADE_EMPTY(bb)) {
        return APR_SUCCESS;
    }
    ...
}
```
A bucket brigade is a doubly-linked list of buckets. The list is terminated (at both ends) by a sentinel which can be distinguished from a normal bucket by comparing it with the pointer returned by APR_BRIGADE_SENTINEL. The list sentinel is in fact not a valid bucket structure; any attempt to call normal bucket functions (such as apr_bucket_read) on the sentinel will have undefined behaviour (i.e. will crash the process).

There are a variety of functions and macros for traversing and manipulating bucket brigades; see the apr_buckets.h header for complete coverage. Commonly used macros include:

**APR_BRIGADE_FIRST(bb)**
- returns the first bucket in brigade bb

**APR_BRIGADE_LAST(bb)**
- returns the last bucket in brigade bb

**APR_BUCKET_NEXT(e)**
- gives the next bucket after bucket e

**APR_BUCKET_PREV(e)**
- gives the bucket before bucket e

The apr_bucket_brigade structure itself is allocated out of a pool, so if a filter creates a new brigade, it must ensure that memory use is correctly bounded. A filter which allocates a new brigade out of the request pool (r->pool) on every invocation, for example, will fall foul of the warning above concerning memory use. Such a filter should instead create a brigade on the first invocation per request, and store that brigade in its state structure.

It is generally never advisable to use apr_brigade_destroy to "destroy" a brigade unless you know for certain that the brigade will never be used again, even then, it should be used
rarely. The memory used by the brigade structure will not be released by calling this function (since it comes from a pool), but the associated pool cleanup is unregistered. Using apr_brigade_destroy can in fact cause memory leaks; if a "destroyed" brigade contains buckets when its containing pool is destroyed, those buckets will not be immediately destroyed.

In general, filters should use apr_brigade_cleanup in preference to apr_brigade_destroy.
When dealing with non-metadata buckets, it is important to understand that the "apr_bucket *" object is an abstract representation of data:

1. The amount of data represented by the bucket may or may not have a determinate length; for a bucket which represents data of indeterminate length, the ->length field is set to the value (apr_size_t)-1. For example, buckets of the PIPE bucket type have an indeterminate length; they represent the output from a pipe.

2. The data represented by a bucket may or may not be mapped into memory. The FILE bucket type, for example, represents data stored in a file on disk.

Filters read the data from a bucket using the apr_bucket_read function. When this function is invoked, the bucket may morph into a different bucket type, and may also insert a new bucket into the bucket brigade. This must happen for buckets which represent data not mapped into memory.

To give an example; consider a bucket brigade containing a single FILE bucket representing an entire file, 24 kilobytes in size:

\[
\text{FILE}(0K-24K)
\]

When this bucket is read, it will read a block of data from the file, morph into a HEAP bucket to represent that data, and return the data to the caller. It also inserts a new FILE bucket representing the remainder of the file; after the apr_bucket_read call, the brigade looks like:

\[
\text{HEAP}(8K) \text{ FILE}(8K-24K)
\]
The basic function of any output filter will be to iterate through the passed-in brigade and transform (or simply examine) the content in some manner. The implementation of the iteration loop is critical to producing a well-behaved output filter.

Taking an example which loops through the entire brigade as follows:

**Bad output filter -- do not imitate!**

```c
apr_bucket *e = APR_BRIGADE_FIRST(bb);
const char *data;
apr_size_t length;

while (e != APR_BRIGADE_SENTINEL(bb)) {
    apr_bucket_read(e, &data, &length, APR_BLOCK_READ);
    e = APR_BUCKET_NEXT(e);
}
return ap_pass_brigade(bb);
```

The above implementation would consume memory proportional to content size. If passed a FILE bucket, for example, the entire file contents would be read into memory as each `apr_bucket_read` call morphed a FILE bucket into a HEAP bucket.

In contrast, the implementation below will consume a fixed amount of memory to filter any brigade; a temporary brigade is needed and must be allocated only once per response, see the **Maintaining state** section.

**Better output filter**

```c
apr_bucket *e;
const char *data;
apr_size_t length;

while ((e = APR_BRIGADE_FIRST(bb)) != APR_BRIGADE_SENTINEL(bb)) {
    ...
}
```
rv = apr_bucket_read(e, &data, &length, APR_BLOCK_READ);
if (rv) ...;
/* Remove bucket e from bb. */
APR_BUCKET_REMOVE(e);
/* Insert it into temporary brigade. */
APR_BRIGADE_INSERT_HEAD(tmpbb, e);
/* Pass brigade downstream. */
rv = ap_pass_brigade(f->next, tmpbb);
if (rv) ...;
apr_brigade_cleanup(tmpbb);
A filter which needs to maintain state over multiple invocations per response can use the ->ctx field of its ap_filter_t structure. It is typical to store a temporary brigade in such a structure, to avoid having to allocate a new brigade per invocation as described in the Brigade structure section.

Example code to maintain filter state

```c
struct dummy_state {
    apr_bucket_brigade *tmpbb;
    int filter_state;
    ...
};
apr_status_t dummy_filter(ap_filter_t *f, apr_bucket_brigade *bb) {
    struct dummy_state *state;
    state = f->ctx;
    if (state == NULL) {
        /* First invocation for this response: initialise state */
        f->ctx = state = apr_palloc(f->r->pool, sizeof *state);
        state->tmpbb = apr_brigade_create(f->r->pool, f->c->bucket_alloc);
        state->filter_state = ...
    }
    ...
}
```
If a filter decides to store buckets beyond the duration of a single filter function invocation (for example storing them in its ->ctx state structure), those buckets must be set aside. This is necessary because some bucket types provide buckets which represent temporary resources (such as stack memory) which will fall out of scope as soon as the filter chain completes processing the brigade.

To setaside a bucket, the apr_bucket_setaside function can be called. Not all bucket types can be setaside, but if successful, the bucket will have morphed to ensure it has a lifetime at least as long as the pool given as an argument to the apr_bucket_setaside function.

Alternatively, the ap_save_brigade function can be used, which will move all the buckets into a separate brigade containing buckets with a lifetime as long as the given pool argument. This function must be used with care, taking into account the following points:

1. On return, ap_save_brigade guarantees that all the buckets in the returned brigade will represent data mapped into memory. If given an input brigade containing, for example, a PIPE bucket, ap_save_brigade will consume an arbitrary amount of memory to store the entire output of the pipe.

2. When ap_save_brigade reads from buckets which cannot be setaside, it will always perform blocking reads, removing the opportunity to use Non-blocking bucket reads.

3. If ap_save_brigade is used without passing a non-NULL "saveto" (destination) brigade parameter, the function will create a new brigade, which may cause memory use to be
proportional to content size as described in the Brigade structure section.

Filters must ensure that any buffered data is processed and passed down the filter chain during the last invocation for a given response (a brigade containing an EOS bucket). Otherwise such data will be lost.
The `apr_bucket_read` function takes an `apr_read_type_e` argument which determines whether a *blocking* or *non-blocking* read will be performed from the data source. A good filter will first attempt to read from every data bucket using a non-blocking read; if that fails with `APR_EAGAIN`, then send a FLUSH bucket down the filter chain, and retry using a blocking read.

This mode of operation ensures that any filters further down the filter chain will flush any buffered buckets if a slow content source is being used.

A CGI script is an example of a slow content source which is implemented as a bucket type. `mod_cgi` will send PIPE buckets which represent the output from a CGI script; reading from such a bucket will block when waiting for the CGI script to produce more output.

**Example code using non-blocking bucket reads**

```c
apr_bucket *e;
apr_read_type_e mode = APR_NONBLOCK_READ;

while ((e = APR_BRIGADE_FIRST(bb)) != APR_BRIGADE_SENTINEL(bb))
    apr_status_t rv;
    rv = apr_bucket_read(e, &data, &length, mode);
    if (rv == APR_EAGAIN && mode == APR_NONBLOCK_READ) {
        /* Pass down a brigade containing a flush bucket: */
        APR_BRIGADE_INSERT_TAIL(tmpbb, apr_bucket_flush_create);
        rv = ap_pass_brigade(f->next, tmpbb);
        apr_brigade_cleanup(tmpbb);
        if (rv != APR_SUCCESS) return rv;

        /* Retry, using a blocking read. */
        mode = APR_BLOCK_READ;
        continue;
    } else if (rv != APR_SUCCESS) {
        /* handle errors */
    }
```
/* Next time, try a non-blocking read first. */
mode = APR_NONBLOCK_READ;
...
In summary, here is a set of rules for all output filters to follow:

1. Output filters should not pass empty brigades down the filter chain, but should be tolerant of being passed empty brigades.
2. Output filters must pass all metadata buckets down the filter chain; FLUSH buckets should be respected by passing any pending or buffered buckets down the filter chain.
3. Output filters should ignore any buckets following an EOS bucket.
4. Output filters must process a fixed amount of data at a time, to ensure that memory consumption is not proportional to the size of the content being filtered.
5. Output filters should be agnostic with respect to bucket types, and must be able to process buckets of unfamiliar type.
6. After calling ap_pass_brigade to pass a brigade down the filter chain, output filters should call apr_brigade_cleanup to ensure the brigade is empty before reusing that brigade structure; output filters should never use apr_brigade_destroy to "destroy" brigades.
7. Output filters must setaside any buckets which are preserved beyond the duration of the filter function.
8. Output filters must not ignore the return value of ap_pass_brigade, and must return appropriate errors back up the filter chain.
9. Output filters must only create a fixed number of bucket brigades for each response, rather than one per invocation.
10. Output filters should first attempt non-blocking reads from each data bucket, and send a FLUSH bucket down the filter chain if the read blocks, before retrying with a blocking read.
Apache HTTP Server 2.x Thread Safety Issues

When using any of the threaded mpms in the Apache HTTP Server 2.x it is important that every function called from Apache be thread safe. When linking in 3rd party extensions it can be difficult to determine whether the resulting server will be thread safe. Casual testing generally won't tell you this either as thread safety problems can lead to subtle race conditions that may only show up in certain conditions under heavy load.
When writing your module or when trying to determine if a module or 3rd party library is thread safe there are some common things to keep in mind.

First, you need to recognize that in a threaded model each individual thread has its own program counter, stack and registers. Local variables live on the stack, so those are fine. You need to watch out for any static or global variables. This doesn't mean that you are absolutely not allowed to use static or global variables. There are times when you actually want something to affect all threads, but generally you need to avoid using them if you want your code to be thread safe.

In the case where you have a global variable that needs to be global and accessed by all threads, be very careful when you update it. If, for example, it is an incrementing counter, you need to atomically increment it to avoid race conditions with other threads. You do this using a mutex (mutual exclusion). Lock the mutex, read the current value, increment it and write it back and then unlock the mutex. Any other thread that wants to modify the value has to first check the mutex and block until it is cleared.

If you are using APR, have a look at the apr_atomic_* functions and the apr_thread_mutex_* functions.
This is a common global variable that holds the error number of the last error that occurred. If one thread calls a low-level function that sets errno and then another thread checks it, we are bleeding error numbers from one thread into another. To solve this, make sure your module or library defines \_REENTRANT or is compiled with -D\_REENTRANT. This will make errno a per-thread variable and should hopefully be transparent to the code. It does this by doing something like this:

```c
#define errno (*(__errno_location()))
```

which means that accessing errno will call __errno_location() which is provided by the libc. Setting \_REENTRANT also forces redefinition of some other functions to their \_*r equivalents and sometimes changes the common getc/putc macros into safer function calls. Check your libc documentation for specifics. Instead of, or in addition to \_REENTRANT the symbols that may affect this are \_POSIX_C\_SOURCE, \_THREAD\_SAFE, \_SVID\_SOURCE, and \_BSD\_SOURCE.
Not only do things have to be thread safe, but they also have to be reentrant. `strtok()` is an obvious one. You call it the first time with your delimiter which it then remembers and on each subsequent call it returns the next token. Obviously if multiple threads are calling it you will have a problem. Most systems have a reentrant version of the function called `strtok_r()` where you pass in an extra argument which contains an allocated char * which the function will use instead of its own static storage for maintaining the tokenizing state. If you are using APR you can use `apr_strtok()`.

crypt() is another function that tends to not be reentrant, so if you run across calls to that function in a library, watch out. On some systems it is reentrant though, so it is not always a problem. If your system has `crypt_r()` chances are you should be using that, or if possible simply avoid the whole mess by using md5 instead.
The following is a list of common libraries that are used by 3rd party Apache modules. You can check to see if your module is using a potentially unsafe library by using tools such as `ldd(1)` and `nm(1)`. For PHP, for example, try this:

```
% ldd libphp4.so
libsablot.so.0 => /usr/local/lib/libsablot.so.0 (0x401f6000)
libexpat.so.0 => /usr/lib/libexpat.so.0 (0x402da000)
libsnmp.so.0 => /usr/lib/libsnmp.so.0 (0x402f9000)
libpdf.so.1 => /usr/local/lib/libpdf.so.1 (0x40353000)
libz.so.1 => /usr/lib/libz.so.1 (0x403e2000)
libpng.so.2 => /usr/lib/libpng.so.2 (0x403f0000)
libmysqlclient.so.11 => /usr/lib/libmysqlclient.so.11 (0x40411000)
libming.so => /usr/lib/libming.so (0x40449000)
libm.so.6 => /lib/libm.so.6 (0x40487000)
libfreetype.so.6 => /usr/lib/libfreetype.so.6 (0x404a8000)
libjpeg.so.62 => /usr/lib/libjpeg.so.62 (0x404e7000)
libcrypt.so.1 => /lib/libcrypt.so.1 (0x40505000)
libssl.so.2 => /lib/libssl.so.2 (0x40532000)
libcrypto.so.2 => /lib/libcrypto.so.2 (0x40560000)
libresolv.so.2 => /lib/libresolv.so.2 (0x40624000)
libdl.so.2 => /lib/libdl.so.2 (0x40634000)
libnsl.so.1 => /lib/libnsl.so.1 (0x40637000)
libc.so.6 => /lib/libc.so.6 (0x4064b000)
/lib/ld-linux.so.2 => /lib/ld-linux.so.2 (0x80000000)
```

In addition to these libraries you will need to have a look at any libraries linked statically into the module. You can use `nm(1)` to look for individual symbols in the module.
Please drop a note to dev@httpd.apache.org if you have additions or corrections to this list.

<table>
<thead>
<tr>
<th>Library</th>
<th>Version</th>
<th>Thread Safe?</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ASpell/PSpell</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Berkeley DB</td>
<td>3.x, 4.x</td>
<td>Yes</td>
<td>Be careful about sharing a connection across threads.</td>
</tr>
<tr>
<td>bzip2</td>
<td></td>
<td>Yes</td>
<td>Both low-level and high-level APIs are thread-safe. However, high-level API requires thread-safe access to errno.</td>
</tr>
<tr>
<td>cdb</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C-Client</td>
<td></td>
<td>Perhaps</td>
<td>c-client uses strtok() and gethostbyname() are not thread-safe on most C library implementations. If strtok() and gethostbyname are thread-safe on your OS, c-client may be thread-safe.</td>
</tr>
<tr>
<td>libcrypt</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expat</td>
<td></td>
<td>Yes</td>
<td>Need a separate parser instance.</td>
</tr>
<tr>
<td>FreeTDS</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>FreeType</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD 1.8.x</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GD 2.0.x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>gdbm</td>
<td>?</td>
<td>No</td>
<td>Errors returned via a static gdbm_error function.</td>
</tr>
<tr>
<td>ImageMagick</td>
<td>5.2.2</td>
<td>Yes</td>
<td>ImageMagick docs claim it is thread-safe since version 5.2.2 (see Change log).</td>
</tr>
<tr>
<td>Imlib2</td>
<td>?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>libjpeg</td>
<td>v6b</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>libmysqlclient</td>
<td></td>
<td>Yes</td>
<td>Use mysqlclient_r library variant.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>For more information, please read <a href="http://dev.mysql.com/doc/mysql/en/Threaded_clients.html">http://dev.mysql.com/doc/mysql/en/Threaded_clients.html</a></td>
</tr>
<tr>
<td>Package</td>
<td>Version</td>
<td>Thread-Safe</td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>Ming</td>
<td>0.2a</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>Net-SNMP</td>
<td>5.0.x</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>OpenLDAP</td>
<td>2.1.x</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>OpenSSL</td>
<td>0.9.6g</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>liboci8 (Oracle 8+)</td>
<td>8.x,9.x</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>pdflib</td>
<td>5.0.x</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>libpng</td>
<td>1.0.x</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>libpng</td>
<td>1.2.x</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>libpq (PostgreSQL)</td>
<td>8.x</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Sablotron</td>
<td>0.95</td>
<td>?</td>
<td></td>
</tr>
<tr>
<td>zlib</td>
<td>1.1.4</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

*Use ldap_r library variant to ensure OpenSSL 0.9.6g.*

*Requires proper usage of CRYPTO_set_locking_callback and CRYPTO_set_id_callback.*

*PDFLib docs claim it is thread safe, it has been partially thread-safe since V1.91: [http://www.pdflib.com/products/pdflib/](http://www.pdflib.com/products/pdflib/).*

*Don't share connections across threads and watch out for crypt() calls.*

*Relies upon thread-safe zalloc and zfree functions it is to use libc's calloc/free which is thread safe.*