Apache Core Features

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

Topics

Directives

AcceptPathInfo Directive

Description:	Resources accept trailing pathname information
Syntax:	`AcceptPathInfo On|Off|Default`
Default:	`AcceptPathInfo Default`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Available in Apache 2.0.30 and later

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

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

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

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

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

`Default`
The treatment of requests with trailing pathname information is determined by the handler [1] responsible for the request. The core handler for normal files defaults to rejecting `PATH_INFO`. Handlers that serve scripts, such as cgi-script [2] and isapi-isa [3], generally accept `PATH_INFO` by default.

The primary purpose of the `AcceptPathInfo` directive is to allow you to override the handler's choice of accepting or rejecting `PATH_INFO`. This override is required, for example, when you use a filter [4], such as INCLUDES [5], to generate content based on `PATH_INFO`. The core handler would usually reject the request, so you can use the following configuration to enable such a script:

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

AccessFileName Directive

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

When returning a document to the client the server looks for the first existing configuration file from this list of names in every directory of the path to the document, if distributed configuration files are enabled for that directory. For example:

```
AccessFileName .acl
```

before returning the document `/usr/local/web/index.html`, the server will read `/.acl`, `/usr/.acl`, `/usr/local/.acl` and `/usr/local/web/.acl` for directives, unless they have been disabled with

```
<Directory />
```

```
  AllowOverride None
</Directory>
```

See also

- AllowOverride
- Configuration Files[6]
- .htaccess Files[7]

AddDefaultCharset Directive

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

This directive specifies the name of the character set that will be added to any response that does not have any parameter on the content type in the HTTP headers. This will override any character set specified in the body of the document via a `META` tag. A setting of `AddDefaultCharset Off` disables this functionality. `AddDefaultCharset On` enables Apache's internal default charset of `iso-8859-1` as required by the directive. You can also specify an alternate *charset* to be used. For example:

```
AddDefaultCharset utf-8
```

AddOutputFilterByType Directive

Description:	assigns an output filter to a particular MIME-type
Syntax:	`AddOutputFilterByType` *`filter`*`[;`*`filter`*`...] `*`MIME-type`*` [`*`MIME-type`*`] ...`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Available in Apache 2.0.33 and later

This directive activates a particular output filter[4] for a request depending on the response MIME-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, they have to be separated by

semicolons. It's also possible to use one AddOutputFilterByType directive for each of them.

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 text/html
</Location>
```

Note:

Enabling filters with AddOutputFilterByType may fail partially or completely in some cases. For expample, no filters are applied if the content type falls back to the DefaultType, even if the DefaultType is the same.

However, if you want to make sure, that the filters will be applied, assign the content type to a resource explicitely, for example with AddType or ForceType. Setting the content type within a (non-nph) CGI script is also safe.

The by-type output filters are never applied on proxy requests.

See also

- AddOutputFilter
- SetOutputFilter
- filters[4]

AllowOverride Directive

Description:	Types of directives that are allowed in .htaccess files
Syntax:	AllowOverride All|None|*directive-type* [*directive-type*] ...
Default:	AllowOverride All
Context:	directory
Status:	Core
Module:	core

When the server finds an .htaccess file (as specified by AccessFileName) it needs to know which directives declared in that file can override earlier access information.

Only available in Directory sections

AllowOverride is valid only in <Directory> sections, not in <Location> or <Files> sections.

When this directive is set to None, then .htaccess files are completely ignored. In this case, the server will not even attempt to read .htaccess files in the filesystem.

When this directive is set to All, then any directive which has the .htaccess Context[8] is allowed in .htaccess files.

The *directive-type* can be one of the following groupings of directives.

AuthConfig
Allow use of the authorization directives (`AuthDBMGroupFile`, `AuthDBMUserFile`, `AuthGroupFile`, `AuthName`, `AuthType`, `AuthUserFile`, `Require`, *etc.*).

FileInfo
Allow use of the directives controlling document types (`DefaultType`, `ErrorDocument`, `ForceType`, `LanguagePriority`, `SetHandler`, `SetInputFilter`, `SetOutputFilter`, and `mod_mime` Add* and Remove* directives, *etc.*).

Indexes
Allow use of the directives controlling directory indexing (`AddDescription`, `AddIcon`, `AddIconByEncoding`, `AddIconByType`, `DefaultIcon`, `DirectoryIndex`, `FancyIndexing`, `HeaderName`, `IndexIgnore`, `IndexOptions`, `ReadmeName`, *etc.*).

Limit
Allow use of the directives controlling host access (`Allow`, `Deny` and `Order`).

Options
Allow use of the directives controlling specific directory features (`Options` and `XBitHack`).

Example:

```
AllowOverride AuthConfig Indexes
```

See also

- `AccessFileName`
- Configuration Files[6]
- .htaccess Files[7]

AuthName Directive

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

This directive sets the name of the authorization realm for a directory. This realm is given to the client so that the user knows which username and password to send. `AuthName` takes a single argument; if the realm name contains spaces, it must be enclosed in quotation marks. It must be accompanied by `AuthType` and `Require` directives, and directives such as `AuthUserFile` and `AuthGroupFile` to work.

For example:

```
AuthName "Top Secret"
```

The string provided for the `AuthName` is what will appear in the password dialog provided by most browsers.

See also

- Authentication, Authorization, and Access Control[9]

AuthType Directive

Description:	Type of user authentication
Syntax:	`AuthType Basic|Digest`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Core
Module:	core

This directive selects the type of user authentication for a directory. Only `Basic` and `Digest` are currently implemented. It must be accompanied by `AuthName` and `Require` directives, and directives such as `AuthUserFile` and `AuthGroupFile` to work.

See also

- Authentication, Authorization, and Access Control[9]

CGIMapExtension Directive

Description:	Technique for locating the interpreter for CGI scripts
Syntax:	`CGIMapExtension` *`cgi-path .extension`*
Default:	`None`
Context:	directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	NetWare only

This directive is used to control how Apache finds the interpreter used to run CGI scripts. For example, setting `CGIMapExtension sys:\foo.nlm .foo` will cause all CGI script files with a .foo extension to be passed to the FOO interpreter.

ContentDigest Directive

Description:	Enables the generation of Content-MD5 HTTP Response headers
Syntax:	`ContentDigest On|Off`
Default:	`ContentDigest Off`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Core
Module:	core

This directive enables the generation of `Content-MD5` headers as defined in RFC1864 respectively RFC2068.

MD5 is an algorithm for computing a "message digest" (sometimes called "fingerprint") of arbitrary-length data, with a high degree of confidence that any alterations in the data will be reflected in alterations in the message digest.

The `Content-MD5` header provides an end-to-end message integrity check (MIC) of the entity-body. A proxy or client may check this header for detecting accidental modification of the entity-body in transit. Example header:

```
Content-MD5: AuLb7Dp1rqtRtxz2m9kRpA==
```

Note that this can cause performance problems on your server since the message digest is computed on every request (the values are not cached).

`Content-MD5` is only sent for documents served by the `core`, and not by any module. For example, SSI documents, output from CGI scripts, and byte range responses do not have this header.

DefaultType Directive

Description:	MIME content-type that will be sent if the server cannot determine a type in any other way
Syntax:	`DefaultType` *`MIME-type`*
Default:	`DefaultType text/plain`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

There will be times when the server is asked to provide a document whose type cannot be determined by its MIME types mappings.

The server must inform the client of the content-type of the document, so in the event of an unknown type it uses the `DefaultType`. For example:

```
DefaultType image/gif
```

would be appropriate for a directory which contained many gif images with filenames missing the `.gif` extension.

Note that unlike `ForceType`, this directive is only provides the default mime-type. All other mime-type definitions, including filename extensions, that might identify the media type will override this default.

<Directory> Directive

Description:	Enclose a group of directives that apply only to the named file-system directory and sub-directories
Syntax:	`<Directory` *`directory-path`*`> ... </Directory>`
Context:	server config, virtual host
Status:	Core

Module:	core

<Directory> and </Directory> are used to enclose a group of directives that will apply only to the named directory and sub-directories of that directory. Any directive that is allowed in a directory context may be used. *Directory-path* is either the full path to a directory, or a wild-card string using Unix shell-style matching. In a wild-card string, ? matches any single character, and * matches any sequences of characters. You may also use [] character ranges. None of the wildcards match a `/' character, so <Directory /*/public_html> will not match /home/user/public_html, but <Directory /home/*/public_html> will match. Example:

```
<Directory /usr/local/httpd/htdocs>
  Options Indexes FollowSymLinks
</Directory>
```

Be careful with the *directory-path* arguments: They have to literally match the filesystem path which Apache uses to access the files. Directives applied to a particular <Directory> will not apply to files accessed from that same directory via a different path, such as via different symbolic links.

Extended regular expressions can also be used, with the addition of the ~ character. For example:

```
<Directory ~ "^/www/.*/[0-9]{3}">
```

would match directories in /www/ that consisted of three numbers.

If multiple (non-regular expression) directory sections match the directory (or its parents) containing a document, then the directives are applied in the order of shortest match first, interspersed with the directives from the .htaccess files. For example, with

```
<Directory />
  AllowOverride None
</Directory>

<Directory /home/>
  AllowOverride FileInfo
</Directory>
```

for access to the document /home/web/dir/doc.html the steps are:

- Apply directive AllowOverride None (disabling .htaccess files).
- Apply directive AllowOverride FileInfo (for directory /home/web).
- Apply any FileInfo directives in /home/web/.htaccess

Regular expressions are not considered until after all of the normal sections have been applied. Then all of the regular expressions are tested in the order they appeared in the configuration file. For example, with

```
<Directory ~ abc$>
  # ... directives here ...
</Directory>
```

The regular expression section won't be considered until after all normal <Directory>s and

`.htaccess` files have been applied. Then the regular expression will match on `/home/abc/public_html/abc` and be applied.

Note that the default Apache access for `<Directory />` is `Allow from All`. This means that Apache will serve any file mapped from an URL. It is recommended that you change this with a block such as

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

and then override this for directories you *want* accessible. See the Security Tips[10] page for more details.

The directory sections occur in the `httpd.conf` file. `<Directory>` directives cannot nest, and cannot appear in a `<Limit>` or `<LimitExcept>` section.

See also

- How Directory, Location and Files sections work[11] for an explanation of how these different sections are combined when a request is received

<DirectoryMatch> Directive

Description:	Enclose directives that apply to file-system directories matching a regular expression and their subdirectories
Syntax:	`<DirectoryMatch` *regex*`> ... </DirectoryMatch>`
Context:	server config, virtual host
Status:	Core
Module:	core

`<DirectoryMatch>` and `</DirectoryMatch>` are used to enclose a group of directives which will apply only to the named directory and sub-directories of that directory, the same as `<Directory>`. However, it takes as an argument a regular expression. For example:

```
<DirectoryMatch "^/www/.*/[0-9]{3}">
```

would match directories in `/www/` that consisted of three numbers.

See also

- `<Directory>` for a description of how regular expressions are mixed in with normal `<Directory>`s
- How Directory, Location and Files sections work[11] for an explanation of how these different sections are combined when a request is received

DocumentRoot Directive

Description:	Directory that forms the main document tree visible from the web
Syntax:	`DocumentRoot` *directory-path*
Default:	`DocumentRoot /usr/local/apache/htdocs`

Context:	server config, virtual host
Status:	Core
Module:	core

This directive sets the directory from which `httpd` will serve files. Unless matched by a directive like `Alias`, the server appends the path from the requested URL to the document root to make the path to the document. Example:

```
DocumentRoot /usr/web
```

then an access to `http://www.my.host.com/index.html` refers to `/usr/web/index.html`.

The `DocumentRoot` should be specified without a trailing slash.

See also

- Mapping URLs to Filesystem Location[12]

EnableMMAP Directive

Description:	Use memory-mapping to read files during delivery
Syntax:	`EnableMMAP On|Off`
Default:	`EnableMMAP On`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

This directive controls whether the `httpd` may use memory-mapping if it needs to read the contents of a file during delivery. By default, when the handling of a request requires access to the data within a file -- for example, when delivering a server-parsed file using `mod_include` -- Apache memory-maps the file if the OS supports it.

This memory-mapping sometimes yields a performance improvement. But in some environments, it is better to disable the memory-mapping to prevent operational problems:

- On some multiprocessor systems, memory-mapping can reduce the performance of the `httpd`.
- With an NFS-mounted `DocumentRoot`, the `httpd` may crash due to a segmentation fault if a file is deleted or truncated while the `httpd` has it memory-mapped.

For server configurations that are vulnerable to these problems, you should disable memory-mapping of delivered files by specifying:

```
EnableMMAP Off
```

For NFS mounted files, this feature may be disabled explicitly for the offending files by specifying:

```
<Directory "/path-to-nfs-files"> EnableMMAP Off </Directory>
```

EnableSendfile Directive

Description:	Use the kernel sendfile support to deliver files to the client
Syntax:	`EnableSendfile On|Off`
Default:	`EnableSendfile On`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Available in version 2.0.44 and later

This directive controls whether `httpd` may use the sendfile support from the kernel to transmit file contents to the client. By default, when the handling of a request requires no access to the data within a file -- for example, when delivering a static file -- Apache uses sendfile to deliver the file contents without ever reading the file if the OS supports it.

This sendfile mechanism avoids seperate read and send operations, and buffer allocations. But on some platforms or within some filesystems, it is better to disable this feature to avoid operational problems:

- 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 a network-mounted `DocumentRoot` (e.g., NFS or SMB), the kernel may be unable to serve the network file through its own cache.

For server configurations that are vulnerable to these problems, you should disable this feature by specifying:

```
EnableSendfile Off
```

For NFS or SMB mounted files, this feature may be disabled explicitly for the offending files by specifying:

```
<Directory "/path-to-nfs-files">
EnableSendfile Off
</Directory>
```

ErrorDocument Directive

Description:	What the server will return to the client in case of an error
Syntax:	`ErrorDocument` *`error-code document`*
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Quoting syntax for text messages is different in Apache 2.0

In the event of a problem or error, Apache can be configured to do one of four things,

1. output a simple hardcoded error message
2. output a customized message
3. redirect to a local *URL-path* to handle the problem/error
4. redirect to an external *URL* to handle the problem/error

The first option is the default, while options 2-4 are configured using the `ErrorDocument` directive, which is followed by the HTTP response code and a URL or a message. Apache will sometimes offer additional information regarding the problem/error.

URLs can begin with a slash (/) for local URLs, or be a full URL which the client can resolve. Alternatively, a message can be provided to be displayed by the browser. Examples:

```
ErrorDocument 500 http://foo.example.com/cgi-bin/tester
ErrorDocument 404 /cgi-bin/bad_urls.pl
ErrorDocument 401 /subscription_info.html
ErrorDocument 403 "Sorry can't allow you access today"
```

Note that when you specify an `ErrorDocument` that points to a remote URL (ie. anything with a method such as `http` in front of it), Apache will send a redirect to the client to tell it where to find the document, even if the document ends up being on the same server. This has several implications, the most important being that the client will not receive the original error status code, but instead will receive a redirect status code. This in turn can confuse web robots and other clients which try to determine if a URL is valid using the status code. In addition, if you use a remote URL in an `ErrorDocument 401`, the client will not know to prompt the user for a password since it will not receive the 401 status code. Therefore, **if you use an "ErrorDocument 401" directive then it must refer to a local document.**

Microsoft Internet Explorer (MSIE) will by default ignore server-generated error messages when they are "too small" and substitute its own "friendly" error messages. The size threshold varies depending on the type of error, but in general, if you make your error document greater than 512 bytes, then MSIE will show the server-generated error rather than masking it. More information is available in Microsoft Knowledgebase article Q294807[13].

Prior to version 2.0, messages were indicated by prefixing them with a single unmatched double quote character.

See also

- documentation of customizable responses[14]

ErrorLog Directive

Description:	Location where the server will log errors
Syntax:	`ErrorLog` *`file-path`*`|syslog[:`*`facility`*`]`
Default:	`ErrorLog logs/error_log (Unix) ErrorLog logs/error.log (Windows and OS/2)`
Context:	server config, virtual host
Status:	Core
Module:	core

The `ErrorLog` directive sets the name of the file to which the server will log any errors it encounters. If the *file-path* does not begin with a slash (/) then it is assumed to be relative to the `ServerRoot`.

Example
```
ErrorLog /var/log/httpd/error_log
```

If the *file-path* begins with a pipe (|) then it is assumed to be a command to spawn to handle the error log.

Example
```
ErrorLog "|/usr/local/bin/httpd_errors"
```

Using `syslog` instead of a filename enables logging via syslogd(8) if the system supports it. The default is to use syslog facility `local7`, but you can override this by using the `syslog:`*`facility`* syntax where *facility* can be one of the names usually documented in syslog(1).

Example
```
ErrorLog syslog:user
```

SECURITY: See the security tips[15] document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

See also

- `LogLevel`
- Apache Log Files[16]

FileETag Directive

Description:	File attributes used to create the ETag HTTP response header
Syntax:	`FileETag` *`component`* `...`
Default:	`FileETag INode MTime Size`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

The `FileETag` directive configures the file attributes that are used to create the `ETag` (entity tag) response header field when the document is based on a file. (The `ETag` value is used in cache management to save network bandwidth.) In Apache 1.3.22 and earlier, the `ETag` value was *always* formed from the file's inode, size, and last-modified time (mtime). The `FileETag` directive allows you to choose which of these -- if any -- should be used. The recognized keywords are:

INode
The file's i-node number will be included in the calculation

MTime
The date and time the file was last modified will be included

Size
The number of bytes in the file will be included

All
All available fields will be used. This is equivalent to:

```
FileETag INode MTime Size
```

None
If a document is file-based, no `ETag` field will be included in the response

The `INode`, `MTime`, and `Size` keywords may be prefixed with either `+` or `-`, which allow changes to be made to the default setting inherited from a broader scope. Any keyword appearing without such a prefix immediately and completely cancels the inherited setting.

If a directory's configuration includes `FileETag INode MTime Size`, and a subdirectory's includes `FileETag -INode`, the setting for that subdirectory (which will be inherited by any sub-subdirectories that don't override it) will be equivalent to `FileETag MTime Size`.

<Files> Directive

Description:	Contains directives that apply to matched filenames
Syntax:	`<Files <i>filename</i>> ... </Files>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `<Files>` directive provides for access control by filename. It is comparable to the `Directory` directive and `Location` directives. It should be matched with a `</Files>` directive. The directives given within this section will be applied to any object with a basename (last component of filename) matching the specified filename. `<Files>` sections are processed in the order they appear in the configuration file, after the `<Directory>` sections and `.htaccess` files are read, but before `<Location>` sections. Note that `<Files>` can be nested inside `<Directory>` sections to restrict the portion of the filesystem they apply to.

The *filename* argument should include a filename, or a wild-card string, where `?` matches any single character, and `*` matches any sequences of characters. Extended regular expressions can also be used, with the addition of the ~ character. For example:

```
<Files ~ "\.(gif|jpe?g|png)$">
```

would match most common Internet graphics formats. In Apache 1.3 and later, `<FilesMatch>` is preferred, however.

Note that unlike `<Directory>` and `<Location>` sections, `<Files>` sections can be used inside `.htaccess` files. This allows users to control access to their own files, at a file-by-file level.

See also

- How Directory, Location and Files sections work [11] for an explanation of how these different sections are combined when a request is received

<FilesMatch> Directive

Description:	Contains directives that apply to regular-expression matched filenames
Syntax:	`<FilesMatch *regex*> ... </FilesMatch>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `<FilesMatch>` directive provides for access control by filename, just as the `<Files>` directive does. However, it accepts a regular expression. For example:

```
<FilesMatch "\.(gif|jpe?g|png)$">
```

would match most common Internet graphics formats.

See also

- How Directory, Location and Files sections work [11] for an explanation of how these different sections are combined when a request is received

ForceType Directive

Description:	Forces all matching files to be served with the specified MIME content-type
Syntax:	`ForceType *MIME-type*|none`
Context:	directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Moved to the core in Apache 2.0

When placed into an `.htaccess` file or a `<Directory>`, or `<Location>` or `<Files>` section, this directive forces all matching files to be served with the content type identification given by *MIME-type*. For example, if you had a directory full of GIF files, but did not want to label them all with `.gif`, you might want to use:

```
ForceType image/gif
```

Note that unlike `DefaultType`, this directive overrides all mime-type associations, including filename extensions, that might identify the media type.

You can override any `ForceType` setting by using the value of `none`:

```
# 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>
```

HostnameLookups Directive

Description:	Enables DNS lookups on client IP addresses
Syntax:	`HostnameLookups On|Off|Double`
Default:	`HostnameLookups Off`
Context:	server config, virtual host, directory
Status:	Core
Module:	core

This directive enables DNS lookups so that host names can be logged (and passed to CGIs/SSIs in `REMOTE_HOST`). The value `Double` refers to doing double-reverse DNS. That is, after a reverse lookup is performed, a forward lookup is then performed on that result. At least one of the ip addresses in the forward lookup must match the original address. (In "tcpwrappers" terminology this is called `PARANOID`.)

Regardless of the setting, when `mod_authz_host` is used for controlling access by hostname, a double reverse lookup will be performed. This is necessary for security. Note that the result of this double-reverse isn't generally available unless you set `HostnameLookups Double`. For example, if only `HostnameLookups On` and a request is made to an object that is protected by hostname restrictions, regardless of whether the double-reverse fails or not, CGIs will still be passed the single-reverse result in `REMOTE_HOST`.

The default is `Off` in order to save the network traffic for those sites that don't truly need the reverse lookups done. It is also better for the end users because they don't have to suffer the extra latency that a lookup entails. Heavily loaded sites should leave this directive `Off`, since DNS lookups can take considerable amounts of time. The utility logresolve[17], provided in the */support* directory, can be used to look up host names from logged IP addresses offline.

IdentityCheck Directive

Description:	Enables logging of the RFC1413 identity of the remote user
Syntax:	`IdentityCheck On|Off`
Default:	`IdentityCheck Off`
Context:	server config, virtual host, directory
Status:	Core
Module:	core

This directive enables RFC1413-compliant logging of the remote user name for each connection, where the client machine runs identd or something similar. This information is logged in the access log.

The information should not be trusted in any way except for rudimentary usage tracking.

Note that this can cause serious latency problems accessing your server since every request requires one of these lookups to be performed. When firewalls are involved each lookup might possibly fail and add 30 seconds of latency to each hit. So in general this is not very useful on public servers accessible from the Internet.

<IfDefine> Directive

Description:	Encloses directives that will be processed only if a test is true at startup
Syntax:	`<IfDefine [!]parameter-name> ... </IfDefine>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `<IfDefine test>...</IfDefine>` section is used to mark directives that are conditional. The directives within an `<IfDefine>` section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

The *test* in the `<IfDefine>` section directive can be one of two forms:

- *parameter-name*
- `!`*parameter-name*

In the former case, the directives between the start and end markers are only processed if the parameter named *parameter-name* is defined. The second format reverses the test, and only processes the directives if *parameter-name* is **not** defined.

The *parameter-name* argument is a define as given on the `httpd` command line via `-D`*parameter-* , at the time the server was started.

`<IfDefine>` sections are nest-able, which can be used to implement simple multiple-parameter tests. Example:

```
httpd -DReverseProxy ...

# httpd.conf
<IfDefine ReverseProxy>
  LoadModule rewrite_module modules/mod_rewrite.so
  LoadModule proxy_module modules/libproxy.so
</IfDefine>
```

<IfModule> Directive

Description:	Encloses directives that are processed conditional on the presence or absence of a specific module
Syntax:	`<IfModule [!]module-name> ... </IfModule>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `<IfModule test>...</IfModule>` section is used to mark directives that are conditional on the presence of a specific module. The directives within an `<IfModule>` section are only processed if the *test* is true. If *test* is false, everything between the start and end markers is ignored.

The *test* in the `<IfModule>` section directive can be one of two forms:

- *module name*

- !*module name*

In the former case, the directives between the start and end markers are only processed if the module named *module name* is included in Apache -- either compiled in or dynamically loaded using `LoadModule`. The second format reverses the test, and only processes the directives if *module name* is **not** included.

The *module name* argument is the file name of the module, at the time it was compiled. For example, `mod_rewrite.c`. If a module consists of several source files, use the name of the file containing the string `STANDARD20_MODULE_STUFF`.

`<IfModule>` sections are nest-able, which can be used to implement simple multiple-module tests.

This section should only be used if you need to have one configuration file that works whether or not a specific module is available. In normal operation, directives need not be placed in `<IfModule>` sections.

Include Directive

Description:	Includes other configuration files from within the server configuration files
Syntax:	`Include` *`file-path|directory-path`*
Context:	server config, virtual host, directory
Status:	Core
Module:	core
Compatibility:	Wildcard matching available in 2.0.41 and later

This directive allows inclusion of other configuration files from within the server configuration files.

Shell-style (fnmatch) wildcard characters can be used to include several files at once, in alphabetical order. In addition, if `Include` points to a directory, rather than a file, Apache will read all files in that directory and any subdirectory. But including entire directories is not recommended, because it is easy to accidentally leave temporary files in a directory that can cause `httpd` to fail.

The file path specified may be a fully qualified path (i.e. starting with a slash), or may be relative to the `ServerRoot` directory.

Examples:

```
Include /usr/local/apache2/conf/ssl.conf
Include /usr/local/apache2/conf/vhosts/*.conf
```

Or, providing paths relative to your `ServerRoot` directory:

```
Include conf/ssl.conf
Include conf/vhosts/*.conf
```

Running `apachectl configtest` will give you a list of the files that are being processed during the configuration check:

```
root@host# apachectl configtest
```

```
Processing config file: /usr/local/apache2/conf/ssl.conf
Processing config file: /usr/local/apache2/conf/vhosts/vhost1.conf
Processing config file: /usr/local/apache2/conf/vhosts/vhost2.conf
Syntax OK
```

See also

- apachectl[18]

KeepAlive Directive

Description:	Enables HTTP persistent connections
Syntax:	`KeepAlive On|Off`
Default:	`KeepAlive On`
Context:	server config, virtual host
Status:	Core
Module:	core

The Keep-Alive extension to HTTP/1.0 and the persistent connection feature of HTTP/1.1 provide long-lived HTTP sessions which allow multiple requests to be sent over the same TCP connection. In some cases this has been shown to result in an almost 50% speedup in latency times for HTML documents with many images. To enable Keep-Alive connections in Apache 1.2 and later, set `KeepAlive On`.

For HTTP/1.0 clients, Keep-Alive connections will only be used if they are specifically requested by a client. In addition, a Keep-Alive connection with an HTTP/1.0 client can only be used when the length of the content is known in advance. This implies that dynamic content such as CGI output, SSI pages, and server-generated directory listings will generally not use Keep-Alive connections to HTTP/1.0 clients. For HTTP/1.1 clients, persistent connections are the default unless otherwise specified. If the client requests it, chunked encoding will be used in order to send content of unknown length over persistent connections.

See also

- `MaxKeepAliveRequests`

KeepAliveTimeout Directive

Description:	Amount of time the server will wait for subsequent requests on a persistent connection
Syntax:	`KeepAliveTimeout` *seconds*
Default:	`KeepAliveTimeout 15`
Context:	server config, virtual host
Status:	Core
Module:	core

The number of seconds Apache will wait for a subsequent request before closing the connection. Once a request has been received, the timeout value specified by the `Timeout` directive applies.

Setting `KeepAliveTimeout` to a high value may cause performance problems in heavily loaded servers. The higher the timeout, the more server processes will be kept occupied waiting on

connections with idle clients.

<Limit> Directive

Description:	Restrict enclosed access controls to only certain HTTP methods
Syntax:	`<Limit method [method] ... > ... </Limit>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Access controls are normally effective for **all** access methods, and this is the usual desired behavior. **In the general case, access control directives should not be placed within a `<Limit>` section.**

The purpose of the `<Limit>` directive is to restrict the effect of the access controls to the nominated HTTP methods. For all other methods, the access restrictions that are enclosed in the `<Limit>` bracket **will have no effect**. The following example applies the access control only to the methods `POST`, `PUT`, and `DELETE`, leaving all other methods unprotected:

```
<Limit POST PUT DELETE>
  Require valid-user
</Limit>
```

The method names listed can be one or more of: `GET`, `POST`, `PUT`, `DELETE`, `CONNECT`, `OPTIONS`, `TRACE`, `PATCH`, `PROPFIND`, `PROPPATCH`, `MKCOL`, `COPY`, `MOVE`, `LOCK`, and `UNLOCK`. **The method name is case-sensitive.** If `GET` is used it will also restrict `HEAD` requests.

<LimitExcept> Directive

Description:	Restrict access controls to all HTTP methods except the named ones
Syntax:	`<LimitExcept method [method] ... > ... </LimitExcept>`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

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

For example:

```
<LimitExcept POST GET>
  Require valid-user
<LimitExcept>
```

LimitRequestBody Directive

Description:	Restricts the total size of the HTTP request body sent from the client
Syntax:	`LimitRequestBody` *`bytes`*
Default:	`LimitRequestBody 0`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

This directive specifies the number of *bytes* from 0 (meaning unlimited) to 2147483647 (2GB) that are allowed in a request body. The default value is defined by the compile-time constant `DEFAULT_LIMIT_REQUEST_BODY` (0 as distributed).

The `LimitRequestBody` directive allows the user to set a limit on the allowed size of an HTTP request message body within the context in which the directive is given (server, per-directory, per-file or per-location). If the client request exceeds that limit, the server will return an error response instead of servicing the request. The size of a normal request message body will vary greatly depending on the nature of the resource and the methods allowed on that resource. CGI scripts typically use the message body for passing form information to the server. Implementations of the `PUT` method will require a value at least as large as any representation that the server wishes to accept for that resource.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks.

If, for example, you are permitting file upload to a particular location, and wich to limit the size of the uploaded file to 100K, you might use the following directive:

```
LimitRequestBody 102400
```

LimitRequestFields Directive

Description:	Limits the number of HTTP request header fields that will be accepted from the client
Syntax:	`LimitRequestFields` *`number`*
Default:	`LimitRequestFields 100`
Context:	server config
Status:	Core
Module:	core

Number is an integer from 0 (meaning unlimited) to 32767. The default value is defined by the compile-time constant `DEFAULT_LIMIT_REQUEST_FIELDS` (100 as distributed).

The `LimitRequestFields` directive allows the server administrator to modify the limit on the number of request header fields allowed in an HTTP request. A server needs this value to be larger than the number of fields that a normal client request might include. The number of request header fields used by a client rarely exceeds 20, but this may vary among different client implementations, often depending upon the extent to which a user has configured their browser to support detailed content negotiation. Optional HTTP extensions are often expressed using request header fields.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks. The value should be

increased if normal clients see an error response from the server that indicates too many fields were sent in the request.

For example:

```
LimitRequestFields 50
```

LimitRequestFieldSize Directive

Description:	Limits the size of the HTTP request header allowed from the client
Syntax:	`LimitRequestFieldsize` *`bytes`*
Default:	`LimitRequestFieldsize 8190`
Context:	server config
Status:	Core
Module:	core

This directive specifies the number of *bytes* from 0 to the value of the compile-time constant `DEFAULT_LIMIT_REQUEST_FIELDSIZE` (8190 as distributed) that will be allowed in an HTTP request header.

The `LimitRequestFieldsize` directive allows the server administrator to reduce the limit on the allowed size of an HTTP request header field below the normal input buffer size compiled with the server. A server needs this value to be large enough to hold any one header field from a normal client request. The size of a normal request header field will vary greatly among different client implementations, often depending upon the extent to which a user has configured their browser to support detailed content negotiation.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks.

For example:

```
LimitRequestFieldSize 16380
```

Under normal conditions, the value should not be changed from the default.

LimitRequestLine Directive

Description:	Limit the size of the HTTP request line that will be accepted from the client
Syntax:	`LimitRequestLine` *`bytes`*
Default:	`LimitRequestLine 8190`
Context:	server config
Status:	Core
Module:	core

This directive sets the number of *bytes* from 0 to the value of the compile-time constant `DEFAULT_LIMIT_REQUEST_LINE` (8190 as distributed) that will be allowed on the HTTP request-line.

The `LimitRequestLine` directive allows the server administrator to reduce the limit on the allowed size of a client's HTTP request-line below the normal input buffer size compiled with the server. Since the request-line consists of the HTTP method, URI, and protocol version, the `LimitRequestLine` directive places a restriction on the length of a request-URI allowed for a request on the server. A server needs this value to be large enough to hold any of its resource names, including any information that might be passed in the query part of a `GET` request.

This directive gives the server administrator greater control over abnormal client request behavior, which may be useful for avoiding some forms of denial-of-service attacks.

For example:

```
LimitRequestLine 16380
```

Under normal conditions, the value should not be changed from the default.

LimitXMLRequestBody Directive

Description:	Limits the size of an XML-based request body
Syntax:	`LimitXMLRequestBody` *`number`*
Default:	`LimitXMLRequestBody 1000000`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Limit (in bytes) on maximum size of an XML-based request body. A value of `0` will disable any checking.

Example:

```
LimitXMLRequestBody 0
```

<Location> Directive

Description:	Applies the enclosed directives only to matching URLs
Syntax:	`<Location` *`URL-path|URL`*`> ... </Location>`
Context:	server config, virtual host
Status:	Core
Module:	core

The `<Location>` directive provides for access control by URL. It is similar to the `<Directory>` directive, and starts a subsection which is terminated with a `</Location>` directive. `<Location>` sections are processed in the order they appear in the configuration file, after the `<Directory>` sections and `.htaccess` files are read, and after the `<Files>` sections.

Note that URLs do not have to line up with the filesystem at all, it should be emphasized that `<Location>` operates completely outside the filesystem.

For all origin (non-proxy) requests, the URL to be matched is a URL-path of the form `/path/`. No scheme, hostname, port, or query string may be included. For proxy requests, the URL to be matched is of the form `scheme://servername/path`, and you must include the prefix.

The URL may use wildcards In a wild-card string, `?` matches any single character, and `*` matches any sequences of characters.

Extended regular expressions can also be used, with the addition of the ~ character. For example:

```
<Location ~ "/(extra|special)/data">
```

would match URLs that contained the substring `/extra/data` or `/special/data`. In Apache 1.3 and above, a new directive `<LocationMatch>` exists which behaves identical to the regex version of `<Location>`.

The `<Location>` functionality is especially useful when combined with the `SetHandler` directive. For example, to enable status requests, but allow them only from browsers at `foo.com`, you might use:

```
<Location /status>
  SetHandler server-status
  Order Deny,Allow
  Deny from all
  Allow from .foo.com
</Location>
```

Note about / (slash)

The slash character has special meaning depending on where in a URL it appears. People may be used to its behavior in the filesystem where multiple adjacent slashes are frequently collapsed to a single slash (*i.e.*, `/home///foo` is the same as `/home/foo`). In URL-space this is not necessarily true. The `<LocationMatch>` directive and the regex version of `<Location>` require you to explicitly specify multiple slashes if that is your intention.

For example, `<LocationMatch ^/abc>` would match the request URL `/abc` but not the request URL `//abc`. The (non-regex) `<Location>` directive behaves similarly when used for proxy requests. But when (non-regex) `<Location>` is used for non-proxy requests it will implicitly match multiple slashes with a single slash. For example, if you specify `<Location /abc/def>` and the request is to `/abc//def` then it will match.

See also

- How Directory, Location and Files sections work [11] for an explanation of how these different sections are combined when a request is received

<LocationMatch> Directive

Description:	Applies the enclosed directives only to regular-expression matching URLs
Syntax:	`<LocationMatch` *`regex`*`> ... </LocationMatch>`
Context:	server config, virtual host
Status:	Core
Module:	core

The <LocationMatch> directive provides for access control by URL, in an identical manner to <Location>. However, it takes a regular expression as an argument instead of a simple string. For example:

```
<LocationMatch "/(extra|special)/data">
```

would match URLs that contained the substring /extra/data or /special/data.

See also

- How Directory, Location and Files sections work [11] for an explanation of how these different sections are combined when a request is received

LogLevel Directive

Description:	Controls the verbosity of the ErrorLog
Syntax:	LogLevel *level*
Default:	LogLevel warn
Context:	server config, virtual host
Status:	Core
Module:	core

LogLevel adjusts the verbosity of the messages recorded in the error logs (see ErrorLog directive). The following *level*s are available, in order of decreasing significance:

Level	Description	Example
emerg	Emergencies - system is unusable.	"Child cannot open lock file. Exiting"
alert	Action must be taken immediately.	"getpwuid: couldn't determine user name from uid"
crit	Critical Conditions.	"socket: Failed to get a socket, exiting child"
error	Error conditions.	"Premature end of script headers"
warn	Warning conditions.	"child process 1234 did not exit, sending another SIGHUP"
notice	Normal but significant condition.	"httpd: caught SIGBUS, attempting to dump core in ..."
info	Informational.	"Server seems busy, (you may need to increase StartServers, or Min/MaxSpareServers)..."
debug	Debug-level messages	"Opening config file ..."

When a particular level is specified, messages from all other levels of higher significance will be reported as well. *E.g.*, when LogLevel info is specified, then messages with log levels of notice and warn will also be posted.

Using a level of at least crit is recommended.

For example:

```
LogLevel notice
```

MaxKeepAliveRequests Directive

Description:	Number of requests allowed on a persistent connection
Syntax:	`MaxKeepAliveRequests` *`number`*
Default:	`MaxKeepAliveRequests 100`
Context:	server config, virtual host
Status:	Core
Module:	core

The `MaxKeepAliveRequests` directive limits the number of requests allowed per connection when `KeepAlive` is on. If it is set to `0`, unlimited requests will be allowed. We recommend that this setting be kept to a high value for maximum server performance.

For example:

```
MaxKeepAliveRequests 500
```

NameVirtualHost Directive

Description:	Designates an IP address for name-virtual hosting
Syntax:	`NameVirtualHost` *`addr`*`[:`*`port`*`]`
Context:	server config
Status:	Core
Module:	core

The `NameVirtualHost` directive is a required directive if you want to configure name-based virtual hosts[19].

Although *addr* can be hostname it is recommended that you always use an IP address, e.g.

```
NameVirtualHost 111.22.33.44
```

With the `NameVirtualHost` directive you specify the IP address on which the server will receive requests for the name-based virtual hosts. This will usually be the address to which your name-based virtual host names resolve. In cases where a firewall or other proxy receives the requests and forwards them on a different IP address to the server, you must specify the IP address of the physical interface on the machine which will be servicing the requests. If you have multiple name-based hosts on multiple addresses, repeat the directive for each address.

Note: the "main server" and any `_default_` servers will **never** be served for a request to a `NameVirtualHost` IP Address (unless for some reason you specify `NameVirtualHost` but then don't define any `VirtualHost`s for that address).

Optionally you can specify a port number on which the name-based virtual hosts should be used, e.g.

```
NameVirtualHost 111.22.33.44:8080
```

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

```
NameVirtualHost [fe80::a00:20ff:fea7:ccea]:8080
```

To receive requests on all interfaces, you can use an argument of `*`

```
NameVirtualHost *
```

Argument to `<VirtualHost>` directive

Note that the argument to the `<VirtualHost>` directive must exactly match the argument to the `NameVirtualHost` directive.

```
NameVirtualHost 1.2.3.4
<VirtualHost 1.2.3.4>
# ...
</VirtualHost>
```

See also

- Virtual Hosts documentation[19]

Options Directive

Description:	Configures what features are available in a particular directory
Syntax:	`Options [+|-]option [[+|-]option] ...`
Default:	`Options All`
Context:	server config, virtual host, directory, .htaccess
Override:	Options
Status:	Core
Module:	core

The `Options` directive controls which server features are available in a particular directory.

option can be set to `None`, in which case none of the extra features are enabled, or one or more of the following:

All
All options except for `MultiViews`. This is the default setting.

ExecCGI
Execution of CGI scripts is permitted.

FollowSymLinks
The server will follow symbolic links in this directory.

> Even though the server follows the symlink it does *not* change the pathname used to match against `<Directory>` sections.
>
> Note also, that this option **gets ignored** if set inside a `<Location>` section.

Includes
Server-side includes are permitted.

IncludesNOEXEC

Server-side includes are permitted, but the `#exec cmd` and `#exec cgi` are disabled. It is still possible to `#include virtual` CGI scripts from `ScriptAlias`e'd directories.

Indexes
If a URL which maps to a directory is requested, and the there is no `DirectoryIndex` (*e.g.*, `index.html`) in that directory, then the server will return a formatted listing of the directory.

MultiViews
Content negotiated[20] "MultiViews" are allowed.

SymLinksIfOwnerMatch
The server will only follow symbolic links for which the target file or directory is owned by the same user id as the link.
Note: this option gets ignored if set inside a `<Location>` section.

Normally, if multiple `Options` could apply to a directory, then the most specific one is taken complete; the options are not merged. However if *all* the options on the `Options` directive are preceded by a `+` or `-` symbol, the options are merged. Any options preceded by a `+` are added to the options currently in force, and any options preceded by a `-` are removed from the options currently in force.

For example, without any `+` and `-` symbols:

```
<Directory /web/docs>
  Options Indexes FollowSymLinks
</Directory>

<Directory /web/docs/spec>
  Options Includes
</Directory>
```

then only `Includes` will be set for the `/web/docs/spec` directory. However if the second `Options` directive uses the `+` and `-` symbols:

```
<Directory /web/docs>
  Options Indexes FollowSymLinks
</Directory>

<Directory /web/docs/spec>
  Options +Includes -Indexes
</Directory>
```

then the options `FollowSymLinks` and `Includes` are set for the `/web/docs/spec` directory.

Note: Using `-IncludesNOEXEC` or `-Includes` disables server-side includes completely regardless of the previous setting.

The default in the absence of any other settings is `All`.

Require Directive

Description:	Selects which authenticated users can access a resource
Syntax:	`Require` *`entity-name`* [*`entity-name`*] ...
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Core

Module:	core

This directive selects which authenticated users can access a directory. The allowed syntaxes are:

- Require user *userid* [*userid*] ...

  Only the named users can access the directory.
- Require group *group-name* [*group-name*] ...

  Only users in the named groups can access the directory.
- Require valid-user

  All valid users can access the directory.

`Require` must be accompanied by `AuthName` and `AuthType` directives, and directives such as `AuthUserFile` and `AuthGroupFile` (to define users and groups) in order to work correctly. Example:

```
AuthType Basic
AuthName "Restricted Directory"
AuthUserFile /web/users
AuthGroupFile /web/groups
Require group admin
```

Access controls which are applied in this way are effective for **all** methods. **This is what is normally desired.** If you wish to apply access controls only to specific methods, while leaving other methods unprotected, then place the `Require` statement into a `<Limit>` section.

See also

- `Satisfy`
- `mod_authz_host`

RLimitCPU Directive

Description:	Limits the CPU consumption of processes launched by Apache children
Syntax:	`RLimitCPU` *`number`*`|max [`*`number`*`|max]`
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes and the second parameter sets the maximum resource limit. Either parameter can be a number, or `max` to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as root, or in the initial startup phase.

This applies to processes forked off from Apache children servicing requests, not the Apache children themselves. This includes CGI scripts and SSI exec commands, but not any processes forked off from the Apache parent such as piped logs.

CPU resource limits are expressed in seconds per process.

See also

- RLimitMEM
- RLimitNPROC

RLimitMEM Directive

Description:	Limits the memory consumption of processes launched by Apache children
Syntax:	`RLimitMEM` *`number`*`|max [`*`number`*`|max]`
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes and the second parameter sets the maximum resource limit. Either parameter can be a number, or `max` to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as root, or in the initial startup phase.

This applies to processes forked off from Apache children servicing requests, not the Apache children themselves. This includes CGI scripts and SSI exec commands, but not any processes forked off from the Apache parent such as piped logs.

Memory resource limits are expressed in bytes per process.

See also

- RLimitCPU
- RLimitNPROC

RLimitNPROC Directive

Description:	Limits the number of processes that can be launched by processes launched by Apache children
Syntax:	`RLimitNPROC` *`number`*`|max [`*`number`*`|max]`
Default:	`Unset; uses operating system defaults`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

Takes 1 or 2 parameters. The first parameter sets the soft resource limit for all processes and the second parameter sets the maximum resource limit. Either parameter can be a number, or `max` to indicate to the server that the limit should be set to the maximum allowed by the operating system configuration. Raising the maximum resource limit requires that the server is running as root, or in the initial startup phase.

This applies to processes forked off from Apache children servicing requests, not the Apache children

themselves. This includes CGI scripts and SSI exec commands, but not any processes forked off from the Apache parent such as piped logs.

Process limits control the number of processes per user.

Note: If CGI processes are **not** running under userids other than the web server userid, this directive will limit the number of processes that the server itself can create. Evidence of this situation will be indicated by **`cannot fork`** messages in the error_log.

See also

- `RLimitMEM`
- `RLimitCPU`

Satisfy Directive

Description:	Interaction between host-level access control and user authentication
Syntax:	`Satisfy Any|All`
Default:	`Satisfy All`
Context:	directory, .htaccess
Override:	AuthConfig
Status:	Core
Module:	core

Access policy if both `Allow` and `Require` used. The parameter can be either `All` or `Any`. This directive is only useful if access to a particular area is being restricted by both username/password *and* client host address. In this case the default behavior (`All`) is to require that the client passes the address access restriction *and* enters a valid username and password. With the `Any` option the client will be granted access if they either pass the host restriction or enter a valid username and password. This can be used to password restrict an area, but to let clients from particular addresses in without prompting for a password.

For example, if you wanted to let people on your network have unrestricted access to a portion of your website, but require that people outside of your network provide a password, you could use a configuration similar to the following:

```
Require valid-user
Allow from 192.168.1
Satisfy Any
```

See also

- `Allow`
- `Require`

ScriptInterpreterSource Directive

Description:	Technique for locating the interpreter for CGI scripts
Syntax:	`ScriptInterpreterSource Registry|Registry-Strict|Script`
Default:	`ScriptInterpreterSource Script`
Context:	server config, virtual host, directory, .htaccess

Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Win32 only option Registry-Strict is available in Apache 2.0 and later

This directive is used to control how Apache finds the interpreter used to run CGI scripts. The default technique is to use the interpreter pointed to by the `#!` line in the script.

Setting `ScriptInterpreterSource Registry` will cause the Windows Registry to be searched using the script file extension (e.g., `.pl`) as a search key.

The option `Registry-Strict` which is new in Apache 2.0 does the same as `Registry` but uses a more strict registry search.

ServerAdmin Directive

Description:	Email address that the server includes in error messages sent to the client
Syntax:	`ServerAdmin` *`email-address`*
Context:	server config, virtual host
Status:	Core
Module:	core

The `ServerAdmin` sets the e-mail address that the server includes in any error messages it returns to the client.

It may be worth setting up a dedicated address for this, e.g.

```
ServerAdmin www-admin@foo.example.com
```

as users do not always mention that they are talking about the server!

ServerAlias Directive

Description:	Alternate names for a host used when matching requests to name-virtual hosts
Syntax:	`ServerAlias` *`hostname`* [*`hostname`*] ...
Context:	virtual host
Status:	Core
Module:	core

The `ServerAlias` directive sets the alternate names for a host, for use with name-based virtual hosts [21].

```
<VirtualHost *>
ServerName server.domain.com
ServerAlias server server2.domain.com server2
# ...
</VirtualHost>
```

See also

- Apache Virtual Host documentation[19]

ServerName Directive

Description:	Hostname and port that the server uses to identify itself
Syntax:	`ServerName` *`fully-qualified-domain-name`*`[:`*`port`*`]`
Context:	server config, virtual host
Status:	Core
Module:	core
Compatibility:	In version 2.0, this directive supersedes the functionality of the Port directive from version 1.3.

The `ServerName` directive sets the hostname and port that the server uses to identify itself. This is used when creating redirection URLs. For example, if the name of the machine hosting the webserver is `simple.example.com`, but the machine also has the DNS alias `www.example.com` and you wish the webserver to be so identified, the following directive should be used:

```
ServerName www.example.com:80
```

If no `ServerName` is specified, then the server attempts to deduce the hostname by performing a reverse lookup on the IP address. If no port is specified in the servername, then the server will use the port from the incoming request. For optimal reliability and predictability, you should specify an explicit hostname and port using the `ServerName` directive.

If you are using name-based virtual hosts [21], the `ServerName` inside a `<VirtualHost>` section specifies what hostname must appear in the request's `Host:` header to match this virtual host.

See the description of the `UseCanonicalName` directive for settings which determine whether self-referential URL's (e.g., by the `mod_dir` module) will refer to the specified port, or to the port number given in the client's request.

See also

- DNS Issues[22]
- Apache virtual host documentation[19]
- `UseCanonicalName`
- `NameVirtualHost`
- `ServerAlias`

ServerPath Directive

Description:	Legacy URL pathname for a name-based virtual host that is accessed by an incompatible browser
Syntax:	`ServerPath` *`URL-path`*
Context:	virtual host
Status:	Core
Module:	core

The `ServerPath` directive sets the legacy URL pathname for a host, for use with name-based virtual hosts[19].

See also

- Apache Virtual Host documentation[19]

ServerRoot Directive

Description:	Base directory for the server installation
Syntax:	`ServerRoot` *`directory-path`*
Default:	`ServerRoot /usr/local/apache`
Context:	server config
Status:	Core
Module:	core

The `ServerRoot` directive sets the directory in which the server lives. Typically it will contain the subdirectories `conf/` and `logs/`. Relative paths for other configuration files are taken as relative to this directory.

Example

```
ServerRoot /home/httpd
```

See also

- the `-d` option to `httpd`[23]
- the security tips[15] for information on how to properly set permissions on the ServerRoot

ServerSignature Directive

Description:	Configures the footer on server-generated documents
Syntax:	`ServerSignature On|Off|EMail`
Default:	`ServerSignature Off`
Context:	server config, virtual host, directory, .htaccess
Override:	All
Status:	Core
Module:	core

The `ServerSignature` directive allows the configuration of a trailing footer line under server-generated documents (error messages, `mod_proxy` ftp directory listings, `mod_info` output, ...). The reason why you would want to enable such a footer line is that in a chain of proxies, the user often has no possibility to tell which of the chained servers actually produced a returned error message.
The `Off` setting, which is the default, suppresses the error line (and is therefore compatible with the behavior of Apache-1.2 and below). The `On` setting simply adds a line with the server version number and `ServerName` of the serving virtual host, and the `EMail` setting additionally creates a "mailto:" reference to the `ServerAdmin` of the referenced document.

After version 2.0.44, the details of the server version number presented are controlled by the `ServerTokens` directive.

See also

- ServerTokens

ServerTokens Directive

Description:	Configures the Server HTTP response header
Syntax:	`ServerTokens Major|Minor|Min[imal]|Prod[uctOnly]|OS|Full`
Default:	`ServerTokens Full`
Context:	server config
Status:	Core
Module:	core

This directive controls whether `Server` response header field which is sent back to clients includes a description of the generic OS-type of the server as well as information about compiled-in modules.

`ServerTokens Prod[uctOnly]`
Server sends (*e.g.*): `Server: Apache`

`ServerTokens Major`
Server sends (*e.g.*): `Server: Apache/2`

`ServerTokens Minor`
Server sends (*e.g.*): `Server: Apache/2.0`

`ServerTokens Min[imal]`
Server sends (*e.g.*): `Server: Apache/2.0.41`

`ServerTokens OS`
Server sends (*e.g.*): `Server: Apache/2.0.41 (Unix)`

`ServerTokens Full` (or not specified)
Server sends (*e.g.*): `Server: Apache/2.0.41 (Unix) PHP/4.2.2 MyMod/1.2`

This setting applies to the entire server, and cannot be enabled or disabled on a virtualhost-by-virtualhost basis.

After version 2.0.44, this directive also controls the information presented by the `ServerSignature` directive.

See also

- ServerSignature

SetHandler Directive

Description:	Forces all matching files to be processed by a handler
Syntax:	`SetHandler` *`handler-name`*`|None`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core
Compatibility:	Moved into the core in Apache 2.0

When placed into an `.htaccess` file or a `<Directory>` or `<Location>` section, this directive

forces all matching files to be parsed through the handler[1] given by *handler-name*. For example, if you had a directory you wanted to be parsed entirely as imagemap rule files, regardless of extension, you might put the following into an `.htaccess` file in that directory:

```
SetHandler imap-file
```

Another example: if you wanted to have the server display a status report whenever a URL of `http://servername/status` was called, you might put the following into httpd.conf:

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

See also

- `AddHandler`

SetInputFilter Directive

Description:	Sets the filters that will process client requests and POST input
Syntax:	`SetInputFilter` *`filter`*`[;`*`filter`*`...]`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

The `SetInputFilter` directive sets the filter or filters which will process client requests and POST input when they are received by the server. This is in addition to any filters defined elsewhere, including the `AddInputFilter` directive.

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content.

See also

- Filters[4] documentation

SetOutputFilter Directive

Description:	Sets the filters that will process responses from the server
Syntax:	`SetOutputFilter` *`filter`*`[;`*`filter`*`...]`
Context:	server config, virtual host, directory, .htaccess
Override:	FileInfo
Status:	Core
Module:	core

The `SetOutputFilter` directive sets the filters which will process responses from the server before they are sent to the client. This is in addition to any filters defined elsewhere, including the `AddOutputFilter` directive.

For example, the following configuration will process all files in the `/www/data/` directory for server-side includes.

```
<Directory /www/data/>
  SetOutputFilter INCLUDES
</Directory>
```

If more than one filter is specified, they must be separated by semicolons in the order in which they should process the content.

See also

- Filters[4] documentation

TimeOut Directive

Description:	Amount of time the server will wait for certain events before failing a request
Syntax:	`TimeOut` *number*
Default:	`TimeOut 300`
Context:	server config
Status:	Core
Module:	core

The `TimeOut` directive currently defines the amount of time Apache will wait for three things:

1. The total amount of time it takes to receive a GET request.
2. The amount of time between receipt of TCP packets on a POST or PUT request.
3. The amount of time between ACKs on transmissions of TCP packets in responses.

We plan on making these separately configurable at some point down the road. The timer used to default to 1200 before 1.2, but has been lowered to 300 which is still far more than necessary in most situations. It is not set any lower by default because there may still be odd places in the code where the timer is not reset when a packet is sent.

UseCanonicalName Directive

Description:	Configures how the server determines its own name and port
Syntax:	`UseCanonicalName On|Off|DNS`
Default:	`UseCanonicalName On`
Context:	server config, virtual host, directory
Status:	Core
Module:	core

In many situations Apache must construct a *self-referential* URL -- that is, a URL that refers back to the same server. With `UseCanonicalName on` Apache will use the hostname and port specified in the `ServerName` directive to construct the canonical name for the server. This name is used in all self-referential URLs, and for the values of `SERVER_NAME` and `SERVER_PORT` in CGIs.

With `UseCanonicalName Off` Apache will form self-referential URLs using the hostname and port supplied by the client if any are supplied (otherwise it will use the canonical name, as defined above).

These values are the same that are used to implement name based virtual hosts[21], and are available with the same clients. The CGI variables `SERVER_NAME` and `SERVER_PORT` will be constructed from the client supplied values as well.

An example where this may be useful is on an intranet server where you have users connecting to the machine using short names such as `www`. You'll notice that if the users type a shortname, and a URL which is a directory, such as `http://www/splat`, *without the trailing slash* then Apache will redirect them to `http://www.domain.com/splat/`. If you have authentication enabled, this will cause the user to have to authenticate twice (once for `www` and once again for `www.domain.com` -- see the FAQ on this subject for more information[24]). But if `UseCanonicalName` is set `Off`, then Apache will redirect to `http://www/splat/`.

There is a third option, `UseCanonicalName DNS`, which is intended for use with mass IP-based virtual hosting to support ancient clients that do not provide a `Host:` header. With this option Apache does a reverse DNS lookup on the server IP address that the client connected to in order to work out self-referential URLs.

Warning:

If CGIs make assumptions about the values of `SERVER_NAME` they may be broken by this option. The client is essentially free to give whatever value they want as a hostname. But if the CGI is only using `SERVER_NAME` to construct self-referential URLs then it should be just fine.

See also

- `ServerName`
- `Listen`

<VirtualHost> Directive

Description:	Contains directives that apply only to a specific hostname or IP address
Syntax:	`<VirtualHost addr[:port] [addr[:port]] ...> ... </VirtualHost>`
Context:	server config
Status:	Core
Module:	core

`<VirtualHost>` and `</VirtualHost>` are used to enclose a group of directives that will apply only to a particular virtual host. Any directive that is allowed in a virtual host context may be used. When the server receives a request for a document on a particular virtual host, it uses the configuration directives enclosed in the `<VirtualHost>` section. *Addr* can be:

- The IP address of the virtual host;
- A fully qualified domain name for the IP address of the virtual host;
- The character `*`, which is used only in combination with `NameVirtualHost *` to match all IP addresses; or
- The string `_default_`, which is used only with IP virtual hosting to catch unmatched IP addresses.

Example

```
<VirtualHost 10.1.2.3>
  ServerAdmin webmaster@host.foo.com
```

```
  DocumentRoot /www/docs/host.foo.com
  ServerName host.foo.com
  ErrorLog logs/host.foo.com-error_log
  TransferLog logs/host.foo.com-access_log
</VirtualHost>
```

IPv6 addresses must be specified in square brackets because the optional port number could not be determined otherwise. An IPv6 example is shown below:

```
<VirtualHost [fe80::a00:20ff:fea7:ccea]>
  ServerAdmin webmaster@host.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>
```

Each Virtual Host must correspond to a different IP address, different port number or a different host name for the server, in the former case the server machine must be configured to accept IP packets for multiple addresses. (If the machine does not have multiple network interfaces, then this can be accomplished with the `ifconfig alias` command (if your OS supports it).

Note

The use of `<VirtualHost>` does **not** affect what addresses Apache listens on. You may need to ensure that Apache is listening on the correct addresses using `Listen`.

When using IP-based virtual hosting, the special name `_default_` can be specified in which case this virtual host will match any IP address that is not explicitly listed in another virtual host. In the absence of any `_default_` virtual host the "main" server config, consisting of all those definitions outside any VirtualHost section, is used when no IP-match occurs. (But note that any IP address that matches a `NameVirtualHost` directive will use neither the "main" server config nor the `_default_` virtual host. See the name-based virtual hosting[21] documentation for further details.)

You can specify a `:port` to change the port that is matched. If unspecified then it defaults to the same port as the most recent `Listen` statement of the main server. You may also specify `:*` to match all ports on that address. (This is recommended when used with `_default_`.)

Security

See the security tips[10] document for details on why your security could be compromised if the directory where logfiles are stored is writable by anyone other than the user that starts the server.

See also

- Apache Virtual Host documentation[19]
- Warnings about DNS and Apache[22]
- Setting which addresses and ports Apache uses[25]
- How Directory, Location and Files sections work[11] for an explanation of how these different sections are combined when a request is received

URI References

[1] http://httpd.apache.org/docs-2.1/handler.html

[2] http://httpd.apache.org/docs-2.1/mod/mod_cgi.html
[3] http://httpd.apache.org/docs-2.1/mod/mod_isapi.html
[4] http://httpd.apache.org/docs-2.1/filter.html
[5] http://httpd.apache.org/docs-2.1/mod/mod_include.html
[6] http://httpd.apache.org/docs-2.1/configuring.html
[7] http://httpd.apache.org/docs-2.1/howto/htaccess.html
[8] http://httpd.apache.org/docs-2.1/mod/directive-dict.html#Context
[9] http://httpd.apache.org/docs-2.1/howto/auth.html
[10] http://httpd.apache.org/docs-2.1/misc/security_tips.html
[11] http://httpd.apache.org/docs-2.1/sections.html
[12] http://httpd.apache.org/docs-2.1/urlmapping.html
[13] http://support.microsoft.com/default.aspx?scid=kb;en-us;Q294807
[14] http://httpd.apache.org/docs-2.1/custom-error.html
[15] http://httpd.apache.org/docs-2.1/misc/security_tips.html#serverroot
[16] http://httpd.apache.org/docs-2.1/logs.html
[17] http://httpd.apache.org/docs-2.1/programs/logresolve.html
[18] http://httpd.apache.org/docs-2.1/programs/apachectl.html
[19] http://httpd.apache.org/docs-2.1/vhosts/
[20] http://httpd.apache.org/docs-2.1/content-negotiation.html
[21] http://httpd.apache.org/docs-2.1/vhosts/name-based.html
[22] http://httpd.apache.org/docs-2.1/dns-caveats.html
[23] http://httpd.apache.org/docs-2.1/invoking.html
[24] http://httpd.apache.org/docs/misc/FAQ.html#prompted-twice
[25] http://httpd.apache.org/docs-2.1/bind.html