Configuration Sections

Available Languages: en | fr | ja | ko | tr

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

See also

- Comments

Types of Configuration Section Containers

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

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

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

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

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

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

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

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

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

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

Filesystem, Webspace, and Boolean Expressions

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

Filesystem Containers

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

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

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

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

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

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

Webspace Containers

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

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

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

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

Overlapping Webspace

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

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

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

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

The same is true for the ProxyPass directives:

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

Wildcards and Regular Expressions

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

If even more flexible matching is required, each container has a regular expression (regex) counterpart <DirectoryMatch>, <FilesMatch>, and <LocationMatch> that allow perl-compatible regular expressions to be used in choosing the matches. But see the section below on configuration merging to find out how using regex sections will change how directives are applied.

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

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

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

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

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

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

Boolean expressions

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

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

What to use When

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

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

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

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

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

Nesting of sections

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

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

Virtual Hosts

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

Proxy

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

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

What Directives are Allowed?

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

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

How the sections are merged

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

The order of merging is:

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

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

Some important remarks:

- Apart from `<Directory>`, within each group the sections are processed in the order they appear in the configuration files. For example, a request for */foo/bar* will match `<Location "/foo/bar">` and `<Location "/foo">` (group 4 in this case): both sections will be evaluated but in the order they appear in the configuration files.
- `<Directory>` (group 1 above) is processed in the order shortest directory component to longest. For example, `<Directory "/var/web/dir">` will be processed before `<Directory "/var/web/dir/subdir">`.
- If multiple `<Directory>` sections apply to the same directory they are processed in the configuration file order.
- Configurations included via the `Include` directive will be treated as if they were inside the including file at the location of the `Include` directive.
- Sections inside `<VirtualHost>` sections are applied *after* the corresponding sections outside the virtual host definition. This allows virtual hosts to override the main server configuration.
- When the request is served by `mod_proxy`, the `<Proxy>` container takes the place of the `<Directory>` container in the processing order.

Technical Note

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

Relationship between modules and configuration sections

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

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

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

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

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

```
</Directory>

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

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

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

Some useful examples

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

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

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

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

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

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

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

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

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

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