

Using Dynamic Control Files

Dynamic control files allow you to change the behaviour of the server while it is running, by altering simple text files stored within the document tree. This allows server configuration to be tailored for each directory, and can be used to delegate server control to those responsible for the content.

Dynamic control files are plain text files which store the configuration details for segments of the document tree. The files contain server directives, followed by optional parameters. White space characters are used as separators and each directive should start a new line. Everything which follows a "#" character is regarded as a comment and is ignored by the server. Directives can be nested unless otherwise stated.

By default, dynamic configuration files use the name ".htaccess" although this is configurable.

Global Dynamic Configuration File

The global dynamic configuration file is treated as a special case. It can specify the settings for any file, directory or URL, associated with the server. The server will apply the configuration modifications defined in the global file to all virtual server requests. These settings can either be absolute or can be overridden by local dynamic configuration files, depending on the configuration. The global dynamic control file should reside in a location outside of the document root, the location can be set from the administration server.

Local Dynamic Configuration Files

Local dynamic configuration files can be placed in any directory within the document root. The server will apply the modifications defined in the file to all requests made to the directory and to any sub directory. This allows the behaviour of the server to be changed for whole segments of the document tree by editing a single file. On a request for the document /products/toys/index.html from a site with a docroot of /pages the server will search :

```
global_htaccess
/.htaccess
/pages/.htaccess
/pages/products/.htaccess
/pages/products/toys/.htaccess
```

It should be noted that the effect of access files are not cumulative, so if access restrictions are relaxed in a protected sub directory, a client will be able to access information if they know the full URL. Be aware that by default all of the directories above the document root on the filesystem are also searched. See the section on [AllowOverride](#) for more information on the search path and how to restrict it.

Macro Expansion

When an htaccess file is dynamically parsed, any instance of the following '%' macro is replaced by its substitution:

Macro	Substitution
-------	--------------

Section Directives

Section directives allow the behaviour of the server to be changed based on external variables and for particular files. Section directives are used in pairs, similar in concept to HTML tags. Each directive has an opening tag, and a closing tag. Any configuration directives placed inside the tags are applied only if the section directives criteria is satisfied. The criteria differs for each section directive. The section directives can be negated by prefixing a "!" to the directive name.

Example.

```
<Section_Directive criteria criteria criteria ...>
    Configuration Directive
</Section_Directive>
```

Example negated.

```
<!Section_Directive criteria criteria criteria ...>
    Configuration Directive
</Section_Directive>
```

Wildcards and Extended Regular Expressions

All the section directives except `<limit>` can make use of wildcards and regular expressions to define their criteria. Wildcards allow simple pattern matching while regular expressions allow more powerful and complex pattern matching. For more details of using [wildcards and extended regular expressions](#) see later sections.

Nesting Section Directives

You can include pairs of section directives within other sections. This allows even finer control over what documents the server sends where. Each section can be nested in any other section, but a section can not be nested in itself (after all it would be a little pointless). If you do nest sections directives you should be aware of the following.

The inner section directive will only be evaluated if the outer directive is evaluated as true. This might appear obvious, but should be considered when ordering directives. The directive with the least processing overhead and occurs most often should be placed first.

If an error occurs in the section directive name, the server will regard the line as an error and ignore it. This will cause all the directives inside the section to be applied every time, regardless of the directive criteria. If an error occurs in the section directive parameters or arguments, the server will recognise the section, but will ignore all enclosed directives until the section close tag is encountered. This behaviour will also apply outside nested sections, but nesting can make the errors more subtle. You should always fully test dynamic configuration files before deployment.

The Section Directives

Directory

The `<Directory>` directive allows the server behaviour to be set for individual directories on the server. The `<Directory>` directive can only be used in the global dynamic configuration file.

Syntax : `<Directory pathspec>`

The `pathspec` argument should be a valid directory name accessible by the server, a valid wild card string or regular expression.

Location

The `<Location>` directive allows the server behaviour to be set for individual server URLs. The `<Location>` directive can only be used in the global dynamic configuration file.

Syntax : `<Location pathspec>`

The `pathspec` argument should be a valid URL for the server, specified without the "http://servername", but starting with a "/", a valid wild card string, or a valid regular expression.

Files

The `<Files>` directive allows the server behaviour to be set for individual files.

Syntax : `<Files filespec>`

The `filespec` argument should be a valid file name within the document tree. If a leading "/" character is not present the server will prefix the current directory to the filename. Alternatively a valid wild card string or valid regular expression.

Limits

The `<limit>` directive allows the server behaviour to be set for individual HTTP methods.

Syntax : `<Limit method method method ...>`

The `method` argument should be a valid HTTP/1.1 method e.g.. GET or POST. It should be in capital letters. Note that "HEAD" requests will be limited in the same way as GETs. HEADs cannot be limited separately.

Mime

The `<Mime>` directive allows the server behaviour to be set for individual MIME types.

Syntax : `<Mime mime-type>`

The `mime-type` argument should be a valid server mime type, a valid wild card string or a valid regular expression.

RemoteIP

The <RemoteIP> directive allows the server behaviour to be set depending on the remote machines IP address.

Syntax : <RemoteIP ipspec>

The ipspec argument should be a valid IP address or subnet, a valid wildcard string or a valid regular expression.

RemoteHost

The <RemoteHost> directive allows the server behaviour to be set depending on the remote machines host name.

Syntax : <RemoteHost hostspec>

The hostspec argument should be a valid host name or domain name, a valid wildcard string or a valid regular expression.

Example

This example shows the structure of dynamic configuration files. Indenting the section directives is not required, but it makes the files easier to read and is therefore good practice.

```
# comments go here

directive parameter
directive parameter

<limit GET POST>
  # more comments here perhaps
  directive parameter
  directive parameter

  <!files *.txt>
    directive parameter
    directive parameter
    directive parameter
  </files>

</limit>
directive parameter
directive parameter
```

Header Directives

Header directives allow HTTP header values to be tailored, modified or overwritten.

Header

The Header directive allows the HTTP response headers returned by the server to be modified. Additional response headers can be inserted, existing response headers modified, deleted and extended.

Syntax :	Header action name value		
Parameters :	action	Set	The header is set, replacing any previous header with this name
		Append	The header is appended on to the existing. The appended value is separated from the existing value by a comma.
		Add	The header is added, even if a header of the name already exists. Can result in multiple headers of the same name.
		Unset	The header value set in parent dynamic configuration files will be unset. No value parameter is required.
	name	The HTTP Header name to insert.	
	value	Text	Literal text value to be associated with the header.
NOW (-/+) offset		Inserts the current server time plus or minus optional offset value in seconds.	

Example

```
Header Append Author MyName
Header set Expires Tue, 17 Jun 1997 18:27:52 GMT

<Mime text/server-parsed-html>
  # All Server side include pages expire in 4 hours time
  Header set Expires NOW+14400
</Mime>
```

AddType

The AddType directive allows additional media types to be set. For the client to correctly handle the document returned by the server, it requires the server to place the correct media type in the HTTP headers. Add type will force any files with a given file extension to return the new media type.

Syntax :	AddType mime-type extension	
Parameters :	mime-type	new media type
	extension	file extension to associate the media type

Example

```
AddType image/jpeg jpg
AddType audio/x-wav wav
AddType video/x-sgi-movie movie
```

ForceType

The ForceType directive allows the media type to be set for every file returned. This is useful when you have a number of files without a file extension, under normal circumstances the server would not know which media type to associate with them.

ForceType will return the specified media type for every file, even ones which may have a valid file extension and associated mime-type.

Syntax :	ForceType mime-type	
Parameters :	mime-type	new media type

Example

```
# All files in this directory are jpegs
ForceType image/jpeg

# All files in specified directory are gifs
<directory /usr/local/web/pictures/landscape/gifs>
  ForceType image/gif
</directory>
```

Redirect

The Redirect directive allows you to inform the client that the resource it has requested is no longer present at the requested location. This may be because it has move or been deleted.

The supplied URL should be used instead. This allows changes in the document tree to be made, without leaving broken links or "404 not found" errors.

Syntax :	Redirect [status] url-path[*] url			
Parameters :	status	"permanent"	301 Moved Permanently	The resource has moved permanently, client should always use the new URL
		"temp"	302 Moved Temporarily	The resource has moved temporarily , client should use the new URL for this request only.
		"see other"	303 See Other	The resource has been replaced, client should use the new URL, but still reference the old resource.
		"gone"	410 Gone	The resource has been permanently removed. The URL parameter should be omitted in this case.
	url-path	absolute path to requested resource		
url	absolute URL to new resource			

When receiving a HTTP 301, 302 or 303 status code the client will automatically request the new URL usually without prompting the user. If the status parameter is omitted, the server will return 303 Moved Temporarily. For a full explanation of these codes refer to the full HTTP1.1 specification.

Examples

```
# redirect to new web site
redirect permanent /products/toys http://www.toys.company.com
# /products is gone forever
redirect gone /products
# no status, use default "move temporary"
redirect /service http://www.company.com/new/service
```

By default, any trailing url info will be appended onto the destination string, however appending a '*' onto the url will force all requests beginning with that url to redirect to exactly the destination, without the trailing url info appended

e.g.

HTaccess line	URL requested	Redirect to
Redirect / http://www.foo.com	/	http://www.foo.com/
	/foo/bar/	http://www.foo.com/foo/bar
Redirect /foo /bar	/foo/bar	/bar/bar
	/fooey	no redirect
Redirect /* http://www.foo.com	/	http://www.foo.com/
	/foo/bar	http://www.foo.com/
Redirect /foo* /bar	/foo/bar	/bar
	/fooey	/bar

Directory Listing Control Directives

These directives allow fine grain control over the output of the directory listing module.

IndexIgnore

The IndexIgnore directive adds to the list of files to hide when listing a directory. File is a leafname, wildcard expression or full filename for files to ignore. Multiple IndexIgnore directives add to the list, rather than the replacing the list of ignored files. By default, the list contains `!`. Example:

```
IndexIgnore .htaccess */*.*? *~ *# */HEADER* */README* */_vti*
```

Error Document Control

These directives allow fine grain control over the error pages generated by the webserver in the event of a problem or error. The customisable error module needs to be enabled for these settings to take effect.

ErrorDocument

The ErrorDocument directive allows you to redirect the client to a local or external url on certain status codes.

Syntax: <ErrorDocument error-code document>

The document can begin with a slash (/) for local URLs, or will be a full URL which the client can resolve. Examples:

```
ErrorDocument 500 /cgi-bin/tester
ErrorDocument 404 /cgi-bin/bad_urls.pl
ErrorDocument 401 http://www2.foo.bar/subscription_info.html
```

chroot Control

These directives allow fine grain control over the selection of the chroot jail.

Chroot

The Chroot directive enables or disables the chroot functionality.

Syntax:

```
Chroot yes|no
```

This value overrides the 'Run/don't run CGI scripts chrooted' setting in the CGI module.

ChrootDir

The ChrootDir directive specifies the location of the chroot jail to be used if chrooting is enabled. You can configure generic chroot jails of the form '%docroot%/my/path' or '%docroot%/../my/path', which allows you to configure individual private (non-shared) jails for multiple subserver web sites.

Example:

```
Chroot yes
ChrootDir %docroot%/../jail
```

... may be a useful setting in a subserver environment where each hosted website has a docroot of /path/to/www.mysite.com/docroot and a preconfigured chroot jail of /path/to/www.mysite.com/jail.

Note that you will also need to configure a [scriptalias](#) to point to the cgi scripts within this jail directory:

```
ScriptAlias cgi-bin %docroot%/../jail/cgi-bin
```

Access Control Directives

Access control allows you to restrict access to information on your web server. This is useful in a number of situations:

- Portions of your web site may be for customers or internal use only.
- Portions may require a subscription to be paid.
- Portions may need to be approved by management before going public.

All these circumstances require some method for the server to validate the client. Zeus Server provides a number of access control methods, providing a balance between ease of administration and performance.

By embedding the access control information within the document structure, whole sections of the web site can be easily locked and unlocked by editing a single file. As most web sites already use directories as a means of organising information into logical hierarchies, dynamic configuration files are ideally suited to the task. By using dynamic control files the access integrity is also increased. If the document structure should change at any time, such as moving or renaming a directory, the access restrictions will still remain in place.

User and Domain Access Control

Access control can be based on two criteria, the user sitting at the machine or the machine itself. This allows you to specify access restrictions which are based on person and on organisation. Host based access control can validate clients using IP number or domain name. Authenticated access control is based on a User ID and Password challenge. Both of the approaches can be used together to allow extra security within an organisation.

Host Based Access Control

Host based access control enables pages to be protected by client location. The server will restrict pages based on either the IP numbers or DNS names listed in the dynamic configuration file. You may want some pages on your server to be accessed only by individuals within your organisation, or customer pages which shouldn't be accessed by the general public. Host based authentication provides a reliable method of restricting access, while keeping administration simple, and the system easy to use.

If the server determines a client request should not be fulfilled, the page requested will not be sent and a "HTTP 403 Forbidden" error will be returned.

Domain Names and IP Numbers

Domain names provide the human readable machine address which we are used to seeing in URLs and email addresses. These names are usually allocated by organisation, so provide a simple means of identifying who is connecting to your web site. Domain names are specified in dynamic configuration files which are specified either absolutely by including the machine name, or sub domain by prefixing a "." to the domain name.

IP numbers are the machine address which the DNS names are mapped onto. They are usually allocated as blocks or subnets as required, one organisation may have a number of IP subnet blocks. This can make restricting pages based on IP number a little more difficult than DNS name.

IP numbers should be listed, as is Internet convention, by converting each of the four bytes to a decimal representation, and separating each byte with a "."s.

IP subnets can be specified in one of three ways.

1. A partial IP-address

For simple class A or B or C subnetting, specify the partial IP address, plus a trailing ".", e.g. 10. to specify the class A 10.255.255.255 network.

2. A network/netmask pair

A.B.C.D/X.Y.W.Z where A.B.C.D is a network, and x.y.w.z is a netmask, e.g.
10.0.0.0/255.0.0.0.

3. A network/n CIDR specification

A.B.C.D/n where A.B.C.D is a network, and n is a number between 1 and 32 specifying the number of high-order 1 bits in the netmask. i.e. 10.0.0/8 is the same as
10.0.0.0/255.0.0.0

Alternatively IP numbers and DNS names can be specified using extended regular expressions. Regular expressions allow sophisticated pattern matching to occur in the dynamic configuration file, but need to be constructed carefully to avoid security holes.

See the Zeus document "Using Regular Expression In Dynamic Configuration Files" for more information.

Examples :

Absolute Names :	ribble.webcom.co.uk
	swale.webcom.co.uk
Sub Domains :	.webcom.co.uk
	.co.uk
Absolute IP Numbers :	194.33.68.245
	194.33.68.134
IP Subnets :	194.33.68.
	194.33.
IP network/netmask pair:	194.33.68.0/255.255.255.0
	194.33.0.0/255.255.0.0
IP network/CIDR representation:	194.33.68.0/8
	194.33.0.0/16

It should be noted that any individual who uses a machine within the specified domain will be permitted to view the restricted pages. Additionally the DNS system has vulnerabilities which allow malicious but technically adept Internet users to "spoof" their DNS names, giving the impression they are within a different domain. Such instances are rare, but it may represent an unacceptable risk for some organisations In which case host based access control should use IP numbers or be augmented by user access control.

Specifying Hosts

Restrictions are specified using the "deny" , "allow" and "order" directives within the dynamic configuration file. The "deny" and "allow" directives take the string "from" followed by either a DNS name or IP address on which to authenticate the client, alternatively the "all" parameter can be used to describe all hosts on the network. You may specify multiple "deny" and "allow" lines within the same dynamic configuration file.

Examples :

```
deny from .webcom.co.uk
deny from red2.ade.co.uk
allow from .ac.uk
allow from .edu
allow from all
deny from 192.241.243.24
deny from 192.241.244.
```

The "order" Directive

To avoid any ambiguity within the "deny" and "allow" lists, the "order" directive specifies which to process first. The order in which the lists are processed have a considerable effect on the restrictions defined. Order can take three values :

deny, allow	Process the "deny" list followed by the "allow" list. Initial state is to allow all. Default behaviour.
allow, deny	Process the "allow" list followed by the "deny" list. Initial state is to deny all.

Creating Usable Policies.

By combining "deny" and "allow" lists sophisticated host authentication can be achieved. Example .htaccess files are listed below for a number of common web site requirements. For the purpose of these examples, the 10.255.255.255 network is considered local.

Deny all access, except internal machines (10.0.0.255 subnet)

```
order deny,allow
deny from all
allow from 10.0.0.
```

Deny all access, except internal machines and our partner company.

```
order deny,allow
deny from all
allow from 10.0.0.
allow from .partner.co.uk
```

Allow all access, except those of our rival.

```
order allow,deny
allow from all
deny from .rival.co.uk
```

Authentication Access Control

Host based access control works well in most situations, particularly when the access to the information should not be made public, but is not sensitive or commercially valuable.

Authenticated access control can offer a greater degree of security by requiring the client to supply a valid username and password before sending the information. It can also be more flexible, allowing authorised users to connect from any machine on the network. User and host based access control can be used together to provide the maximum security option.

Authenticated access control has a number of additional administrative overheads, which make it a little more complicated than host based access control.

How Authenticated Access Control works

When a client tries to connect to a resource which is protected with Authenticated access control the server will return a HTTP Status code value "**401 Unauthorised**" to the client. The client should then display a dialogue box asking for a username and password. The resource is then requested again, this time the client will include the "**Authorised**" HTTP header which includes the username and password. The server then compares them against user lists and if both fields are valid the server will return the resource. If the username or password is incorrect, the server will again return the HTTP Status code value "**401 Unauthorised**" header, where by the client can ask again for the login details.

To access resources which are protected with Authenticated access control the client must provide the login details for each request. If the client prompted for login details for every file it

transferred the process would be slow, tedious and inconvenient. To solve this problem the client will send the "**Authorised**" HTTP header, for each subsequent request from the site.

Authenticated access control is configured in the same manor as host based access control. Directives are included in the dynamic configuration files which specify which users to allow and which resources to protect. These set of Authenticated dynamic configuration directives define a realm, which the server applies to the protected resources.

User Databases

The username and password information for Authenticated access control are stored as plain text files. For security reasons it is important that these files are *not* stored under the document root. The format for these files is similar to the standard UNIX `/etc/passwd` file.

Example :

```
username:encrypted-password
bones:yFMShLmkYQRNs
kirk:Ts/MNg8m1HFEI
pike:DBXD3aA2b.fRI
spock:rVQVg9NORtXww
```

The password is encrypted in the same manor used in `/etc/passwd`, enabling easy manipulation of the user files by third parties. Additional information may be stored in the file by appending an additional ":" to the password field. Any fields following the password entry will be ignored by the server.

```
username:encrypted-password:real-name:bday
```

This does allow the `/etc/passwd` file to be used for Authenticated access control, although again for security reasons this is not recommended. Username and password information is transmitted by the browser unencrypted, malicious individuals could intercept (snoop) the data and obtain the login and password details. Although this is the same for other protocols like telnet, and ftp. The frequency of HTTP requests can increase the risk.

Users are added to the database using the `htpasswd` program. The `htpasswd` program can be found in the `$ZEUSHOME/web/bin` directory, there is also a Windows95 and NT version available from the Zeus website. This program will create the necessary encrypted password entry for the user. `htpasswd` takes an optional parameter, followed by the location of the user database file and the new user name. The optional parameters are `-c` to create a new database file, and `-d` to delete the named user from the specified user database file.

Examples :

```
$ htpasswd
Usage: htpasswd [-(c|d)] <passwdfile>
<username>

$ htpasswd -c /usr/local/etc/webpasswd fred
New Password: *****
Re-type new password: *****

$ htpasswd /usr/local/etc/webpasswd barney
```

```
New Password: *****  
Re-type new password: *****
```

```
$ htpasswd -d /usr/local/etc/webpasswd betty
```

The Password "*" characters are added for clarity, and do not appear when using the program.

Group Files

Group files allow users in the user database files to be grouped logically together. Group files are plain text files, each unique group is stored on an individual line in the file. This is the same format as traditional UNIX group files. Users may be members of multiple groups. Example :

```
groupname:user1 user2 user3  
crew:kirk spock bones  
command:kirk pike
```

Authenticated Dynamic Control Directives

The Authenticated access control directives provide the server with the necessary information to authenticate the client requests.

Authenticated access control is applied to a realm. Realms are used to describe the areas on a site which are protected. A realm will usually take the form of a directory containing the protected resources and an appropriate dynamic configuration file. Once a user has supplied a valid username and password combination for the realm, any other areas of the site which are protected with the same realm details can be accessed with the same username and password. This allows different portions of the document tree to be protected, while only requiring the user to login once. It also allows the client to cache the username and password details if the user should return to the same realm at the same site.

AuthName : Each realm is identified by the AuthName directive. The directive takes a text string parameter which is used as an identifier for the realm. The string is usually displayed by the browser when prompting the user for a username and password.

```
AuthName Realm  
AuthName Company Protected Pages  
AuthName Subscription Service
```

AuthType : Specifies the method by which the client should transmit the username and password to the server. Currently the only supported option is "basic", however another method, called "digest", which offers additional security is currently undergoing approval by the Internet Engineering Task Force.

```
AuthType BASIC
```

Basic authentication instructs the browser to send the password to the server uuencode. This is not plain text, but it is also not encrypted. Basic authentication should be treated with the same security considerations as telnet, but as HTTP requests are far more frequent than telnet logins the chance of having your password "snooped" is increased. It is for this reason which we recommend that you do not use your /etc/passwd file as your user database.

AuthUserFile : Gives the location of the user database file which the server is to use to

authenticate the client. This should be readable by the server and created by the htaccess program. If the filename begins with a '/' it is considered an absolute pathname, otherwise it is relative to the directory in which the .htaccess file lives. In the later case, for extra security you could also restrict people from downloading the password file by using the <file> and deny from all directives.

```
AuthUserFile /usr/local/etc/webusers
AuthUserFile /usr/local/www/passwords
AuthUserFile mypasswd
```

AuthGroupFile : Gives the location of the group file which the server is to apply to the user database. If the filename begins with a '/' it is considered an absolute pathname, otherwise it is relative to the directory in which the .htaccess file lives. In the later case, for extra security you could also restrict people from downloading the group file by using the <file> and deny from all directives.

```
AuthGroupFile /usr/local/etc/webgroups
AuthGroupFile /usr/local/www/departments
```

AuthDBMUserFile : Gives the location of the user database stored in DBM format. The user file is keyed on the username. The value for a user is the crypt() encrypted password, optionally followed by a colon and arbitrary data. The colon and the data following it will be ignored.

AuthDBMGroupFile : Gives the location of the group database stored in DBM format. The group file is keyed on the username. The value for a user is either a comma separated list of groups that user is in, or a value of:

UNIX crypted passwd : Comma separated list of groups [: (ignored)]

Files in the later format can be used for both the user & passwd database.

Require : Takes either a list of group names, or usernames which are allowed to access the protected resource. Groups or users are identified by including the group or user parameter after the require directive.

```
Require user admin
Require group flintstones jetsons
```

Additionally the valid-user parameter can be used to include any valid username password combination.

```
Require valid-user
```

Examples

The following examples will give you an idea of what Zeus dynamic configuration files should look like, and what can be achieved using them. Pages locked to all but valid users.

```
AuthName Locked Web Pages
AuthType basic
AuthUserFile /usr/local/www/userdata
Require valid-user
```

Pages locked to all but users in the webmaster group.

```
AuthName New set of pages
AuthType basic
AuthUserFile /usr/local/www/userdata
AuthGroupFile /usr/local/www/groupdata
Require group webmaster
```

Pages locked to all but users in the management group and the admin user.

```
AuthName Quarterly Sales Figures
AuthType basic
AuthUserFile /usr/local/www/users
AuthGroupFile /usr/local/www/groupdata
Require group management
Require user admin
```

Using Authentication and Access Control Together

You can mix and match different security policies, using access control where needed, authentication where necessary, and both together where required. The more sophisticated policies make use of the `<remoteip>` and `<remotehost>` section directives. These section directives are more flexible than the `allow` and `deny` directives as they can enclose other directives.

Examples

For the purpose of these examples, the 10.255.255.255 network is considered local.

Require all access to be from a local machine and be authenticated

```
Order deny,allow
Deny from all
Allow from 10.0.0.
AuthName Protected
AuthType basic
AuthUserFile /usr/local/www/users
AuthGroupFile /usr/local/www/groupdata
Require valid-user
```

Allow access from all local machines, all external access require authentication

```
<!RemoteIP 10.0.0.*>
  AuthName Protected
  AuthType basic
  AuthUserFile /usr/local/www/users
  AuthGroupFile /usr/local/www/groupdata
  Require Valid-User
</RemoteIP>
```

Allow access to all company machines, all application downloads require authentication

```
<!RemoteHost *.zeus.co.uk>
  <Mime application/*>
    AuthName Password for Download
    AuthType basic
    AuthUserFile /usr/local/www/users
    Require valid-user
  </Mime>
```

```
</RemoteHost>
```

Limit PUT publishing to members of the publishers group, and only from local machines

```
<limit PUT>
  order deny,allow
  deny from all
  allow from 10.0.0.
  AuthName Password for upload
  AuthType basic
  AuthUserFile /usr/local/www/users
  AuthGroupFile /usr/local/www/groupdata
  Require group publishers
</limit>
```

Allow publishing from local machines with password, and from remote machines with a different password.

```
<limit PUT>
  <remoteip 10.0.0.*>
    AuthName Password for upload
    AuthType basic
    AuthUserFile /etc/passwd
    AuthGroupFile /usr/local/www/groupdata
    Require group publishers
  </remoteip>
  <!remoteip 10.0.0.*>
    AuthName Secure Password for upload
    AuthType basic
    AuthUserFile /usr/local/www/securepasswd
    AuthGroupFile /usr/local/www/groupdata
    Require group publishers
  </remoteip>
</limit>
```

Restriction Directives

Restriction directives can be used to limit the servers functionality. It may often be desirable for some facilities to be disabled for portions of the document tree, such as those which contain user home directories.

Options The options directive allows CGI facilities to be disabled.

Syntax :	Options option
Parameters :	all, execcgi Allow CGI programs stored in the directory to be run
	none Disallow CGI programs stored in the directory to be run

Default behaviour is : options all

Example

```
#turn on cgi programs in docroot
options execcgi
#disable them in user home directories
```

```

<directory /home>
  options none
</directory>

```

AllowOverride The allow override directive sets the extent to which dynamic configuration files can change higher level settings. The allowoverride directive can only be used in the global dynamic configuration file.

Syntax :	allowoverride override override ...	
Parameters :	all	Allow override all dynamic configuration directives to be used
	options	Allow use of the options directive
	fileinfo	Allow the addtype and forcetype directive
	authconfig	Allow the user authentication directives, authuserfile, authgroupfile, authtype, authname,
	limit	Allow the <limit> section directive
	none	Disallow all dynamic configuration directives. .htaccess files in these directories will not be looked at.

Default behaviour is : allowoverride all

Setting AllowOverride none prevents the server from looking for .htaccess files in those directories. This can offer a substantial performance advantage, and on busy sites you should disable the server from looking in commonly accessed directories for .htaccess files when those directories will never contain .htaccess directories. For example, if your document root is /disk2/web/www.mysite.com, the webserver will look for htaccess files at:

```

/.htaccess
/disk2/.htaccess
/disk2/web/.htaccess
/disk2/web/www.mysite.com/.htaccess

```

as well as the global htaccess file if set. Most webmasters don't put .htaccess files above the document directory, so as a performance improvement, you can let the server know that it shouldn't look for them outside the document root. E.g.

```

<Directory />
# Prevent the server from looking at .htaccess files anywhere...
AllowOverride none
</Directory>
<Directory /disk2/web/www.mysite.com/>
# ... but get the server to look for them in this subtree.
AllowOverride all
</Directory>

```

Directory Directives

Directory directives can be used to map directories outside on the file system to locations within the document tree. Directory directives are also used to specify the location on the server of extension programs, such as CGI programs or ISAPI modules. Because extension programs can be a potential security hazard, it is often desirable to limit where they can reside. Allowing all users to run extension programs should be discouraged.

Directory directives can only be used in the global configuration file.

Alias The Alias directive maps a location on the file system to a virtual location in the document root.

Syntax :	Alias virtual-dir logical-dir	
Parameters :	virtual-dir	The virtual path name used by the web server
	logical-dir	The absolute path name to the directory on the local file system

Example

```
# Maps ftp images directory to /web/images
alias /web/images /home/ftp/pub/images
```

ScriptAlias The ScriptAlias directive allows CGI programs to reside outside of the document root. Any file in a scriptalias directory is regarded as a CGI program and when accessed will be run by the server. CGI programs in a scriptalias directory do not need a mime type of application/x-httpd-cgi as they would in the docroot.

Scriptalias directories allow you to manage you site more securely. CGI programs run on the same machine as the web server and will consume resources. It is possible that a badly written CGI program could bring down the whole machine by starving it of resources. For this reason it is advised that you only run approved CGI program on your system.

Syntax :	scriptalias virtual-dir logical-dir	
Parameters :	virtual-dir	The virtual path name used by the web server
	logical-dir	The absolute path name to the directory on the local file system

Example

```
# Maps customers approved CGI programs to /cgibin
scriptalias /cgibin /usr/local/web/cgi/approved
# Add local cgi programs to /cgibin
scriptalias /cgibin /usr/local/web/cgi/localscripts
```

ISAPIAlias The isapialias directive allows ISAPI modules to reside outside of the document root. Any file in an isapialias directory is regarded as an ISAPI module and when accessed will be run by the server. ISAPI modules in a isapialias directory do not need a mime type of application/x-httpd-isapi as they would in the docroot.

ISAPI modules run within the same process as the web server and as such have a number of advantages over CGI programs. However a badly written ISAPI module can crash the server process itself. It is advised not to allow isapi modules which you do not have confidence in to run on your system.

Syntax :	isapialias virtual-dir logical-dir	
Parameters :	virtual-dir	The virtual path name used by the web server
	logical-dir	The absolute path name to the directory on the local file system

Example

```
# Maps customers approved
ISAPI modules to /isapi
```

```
isapialias /isapi /usr/local/web/isapi/approved
# Add local isapi programs to /isapi
scriptalias /isapi /usr/local/web/isapi/localscripts
```

Miscellaneous Directives

PassEnvAuthorization

PassEnvAuthorization on|off (*default off*)

Only available in a global htaccess file. Can be embedded in any sectioning directive.

When set to 'on', the environment data for dynamic applications (CGIs/JServ/FastCGI/NSAPI etc) will contain the client's 'Authorization:' header as 'HTTP_AUTHORIZATION'. This allows the application to perform its own access control. e.g.

```
<Location /distributed/>
PassEnvAuthorization on
</Location>
```

Obviously only 'trusted' applications should be given the client's password information for their access control purposes. Application servers such as Zope & some Java Servlets require this information.

PassEnvAuthorization is a Zeus extension to the Apache specification.

Using Wildcards and Regular Expressions

All the section directives except `<limit>` can make use of wildcards and regular expressions to define their criteria. Wildcards allow simple pattern matching while regular expressions allow more powerful and complex pattern matching.

Wildcards

*	Any sequence of characters
?	Any single character

Using wildcards in `<files>` section directive, in an example directory consisting of the following files:

<code><files *></code>	footer.html index.html index2.html logo.gif logo2.gif logo3.gif
<code><files *.html></code>	footer.html index.html index2.html
<code><files logo.*></code>	logo.gif
<code><files logo*></code>	logo.gif logo2.gif logo3.gif
<code><files logo?.gif></code>	logo2.gif

logo3.gif

