Expressions in Apache HTTP Server

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

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

Grammar in Backus-Naur Form notation

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

```
expr        ::= cond
              | string

string      ::= substring
              | string substring

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

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


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

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

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

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

substring   ::= cstring
              | variable

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

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

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

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

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

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

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

    words       ::= word
                | word "," list

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

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

    regany      ::= regex | regsub

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

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

    rebackref   ::= "$" DIGIT

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

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

Variables

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

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

Name
HTTP_ACCEPT
HTTP_COOKIE
HTTP_FORWARDED
HTTP_HOST
HTTP_PROXY_CONNECTION
HTTP_REFERER
HTTP_USER_AGENT

Other request related variables

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

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

Misc variables

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

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

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

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

Binary operators

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

Comparison operators

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

-ge	ge	Integer greater than or equal

Other binary operators

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

Unary operators

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

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

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

Functions

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

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

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

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

Environment variable ordering

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

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

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

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

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

Other

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

Regular expression backreferences

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

Comparison with SSLRequire

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

Version History

The req_novary function is available for versions 2.4.4 and later.

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

Example expressions

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

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

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

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

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

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

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

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

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

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

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

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

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

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