```
                          ----------------------
                                HAProxy
                           Configuration Manual
                          ----------------------
                              version 1.5.15
                              willy tarreau
                                2015/11/01


This document covers the configuration language as implemented in the version
specified above. It does not provide any hint, example or advice. For such
documentation, please refer to the Reference Manual or the Architecture Manual.
The summary below is meant to help you search sections by name and navigate
through the document.

Note to documentation contributors :
    This document is formatted with 80 columns per line, with even number of
    spaces for indentation and without tabs. Please follow these rules strictly
    so that it remains easily printable everywhere. If a line needs to be
    printed verbatim and does not fit, please end each line with a backslash
    ('\') and continue on next line, indented by two characters. It is also
    sometimes useful to prefix all output lines (logs, console outs) with 3
    closing angle brackets ('>>>') in order to help get the difference between
    inputs and outputs when it can become ambiguous. If you add sections,
    please update the summary below for easier searching.


Summary
-------

1.    Quick reminder about HTTP
1.1.      The HTTP transaction model
1.2.      HTTP request
1.2.1.        The Request line
1.2.2.        The request headers
1.3.      HTTP response
1.3.1.        The Response line
1.3.2.        The response headers

2.    Configuring HAProxy
2.1.      Configuration file format
2.2.      Time format
2.3.      Examples

3.    Global parameters
3.1.      Process management and security
3.2.      Performance tuning
3.3.      Debugging
3.4.      Userlists
3.5.      Peers

4.    Proxies
4.1.      Proxy keywords matrix
4.2.      Alphabetically sorted keywords reference

5.    Bind and Server options
5.1.      Bind options
5.2.      Server and default-server options

6.    HTTP header manipulation

7.    Using ACLs and fetching samples
7.1.      ACL basics
7.1.1.      Matching booleans
7.1.2.      Matching integers
```



```
7.1.3.      Matching strings
7.1.4.      Matching regular expressions (regexes)
7.1.5.      Matching arbitrary data blocks
7.1.6.      Matching IPv4 and IPv6 addresses
7.2.      Using ACLs to form conditions
7.3.      Fetching samples
7.3.1.        Converters
7.3.2.        Fetching samples from internal states
7.3.3.        Fetching samples at Layer 4
7.3.4.        Fetching samples at Layer 5
7.3.5.        Fetching samples from buffer contents (Layer 6)
7.3.6.        Fetching HTTP samples (Layer 7)
7.4.      Pre-defined ACLs

8.    Logging
8.1.      Log levels
8.2.      Log formats
8.2.1.        Default log format
8.2.2.        TCP log format
8.2.3.        HTTP log format
8.2.4.        Custom log format
8.2.5.        Error log format
8.3.      Advanced logging options
8.3.1.        Disabling logging of external tests
8.3.2.        Logging before waiting for the session to terminate
8.3.3.        Raising log level upon errors
8.3.4.        Disabling logging of successful connections
8.4.      Timing events
8.5.      Session state at disconnection
8.6.      Non-printable characters
8.7.      Capturing HTTP cookies
8.8.      Capturing HTTP headers
8.9.      Examples of logs

9.    Statistics and monitoring
9.1.      CSV format
9.2.      Unix Socket commands


1. Quick reminder about HTTP
----------------------------

When haproxy is running in HTTP mode, both the request and the response are
fully analyzed and indexed, thus it becomes possible to build matching criteria
on almost anything found in the contents.

However, it is important to understand how HTTP requests and responses are
formed, and how HAProxy decomposes them. It will then become easier to write
correct rules and to debug existing configurations.


1.1. The HTTP transaction model
-------------------------------

The HTTP protocol is transaction-driven. This means that each request will lead
to one and only one response. Traditionally, a TCP connection is established
from the client to the server, a request is sent by the client on the
connection, the server responds and the connection is closed. A new request
will involve a new connection :

  [CON1] [REQ1] ... [RESP1] [CLO1] [CON2] [REQ2] ... [RESP2] [CLO2] ...

In this mode, called the "HTTP close" mode, there are as many connection
establishments as there are HTTP transactions. Since the connection is closed
by the server after the response, the client does not need to know the content
```

length.

Due to the transactional nature of the protocol, it was possible to improve it
to avoid closing a connection between two subsequent transactions. In this mode
however, it is mandatory that the server indicates the content length for each
response so that the client does not wait indefinitely. For this, a special
header is used: "Content-length". This mode is called the "keep-alive" mode :

  [CON] [REQ1] ... [RESP1] [REQ2] ... [RESP2] [CLO] ...

Its advantages are a reduced latency between transactions, and less processing
power required on the server side. It is generally better than the close mode,
but not always because the clients often limit their concurrent connections to
a smaller value.

A last improvement in the communications is the pipelining mode. It still uses
keep-alive, but the client does not wait for the first response to send the
second request. This is useful for fetching large number of images composing a
page :

  [CON] [REQ1] [REQ2] ... [RESP1] [RESP2] [CLO] ...

This can obviously have a tremendous benefit on performance because the network
latency is eliminated between subsequent requests. Many HTTP agents do not
correctly support pipelining since there is no way to associate a response with
the corresponding request in HTTP. For this reason, it is mandatory for the
server to reply in the exact same order as the requests were received.

By default HAProxy operates in keep-alive mode with regards to persistent
connections: for each connection it processes each request and response, and
leaves the connection idle on both sides between the end of a response and the
start of a new request.

HAProxy supports 5 connection modes :
  - keep alive   : all requests and responses are processed (default)
  - tunnel   : only the first request and response are processed,
                    everything else is forwarded with no analysis.
  - passive close : tunnel with "Connection: close" added in both directions.
  - server close : the server-facing connection is closed after the response.
  - forced close : the connection is actively closed after end of response.

1.2. HTTP request

First, let's consider this HTTP request :

  Line   Contents
  number
     1   GET /serv/login.php?lang=en&profile=2 HTTP/1.1
     2   Host: www.mydomain.com
     3   User-agent: my small browser
     4   Accept: image/jpeg, image/gif
     5   Accept: image/png

1.2.1. The Request line

Line 1 is the "request line". It is always composed of 3 fields :

  - a METHOD   : GET
  - a URI   : /serv/login.php?lang=en&profile=2
  - a version tag : HTTP/1.1

All of them are delimited by what the standard calls LWS (linear white spaces),
which are commonly spaces, but can also be tabs or line feeds/carriage returns
followed by spaces/tabs. The method itself cannot contain any colon (':') and
is limited to alphabetic letters. All those various combinations make it
desirable that HAProxy performs the splitting itself rather than leaving it to
the user to write a complex or inaccurate regular expression.

The URI itself can have several forms :

  - A "relative URI" :

      /serv/login.php?lang=en&profile=2

    It is a complete URL without the host part. This is generally what is
    received by servers, reverse proxies and transparent proxies.

  - An "absolute URI", also called a "URL" :

      http://192.168.0.12:8080/serv/login.php?lang=en&profile=2

    It is composed of a "scheme" (the protocol name followed by '://'), a host
    name or address, optionally a colon (':') followed by a port number, then
    a relative URI beginning at the first slash ('/') after the address part.
    This is generally what proxies receive, but a server supporting HTTP/1.1
    must accept this form too.

  - a star ('*') : this form is only accepted in association with the OPTIONS
    method and is not relayable. It is used to inquiry a next hop's
    capabilities.

  - an address:port combination : 192.168.0.12:80
    This is used with the CONNECT method, which is used to establish TCP
    tunnels through HTTP proxies, generally for HTTPS, but sometimes for
    other protocols too.

In a relative URI, two sub-parts are identified. The part before the question
mark is called the "path". It is typically the relative path to static objects
on the server. The part after the question mark is called the "query string".
It is mostly used with GET requests sent to dynamic scripts and is very
specific to the language, framework or application in use.

1.2.2. The request headers

The headers start at the second line. They are composed of a name at the
beginning of the line, immediately followed by a colon (':'). Traditionally,
an LWS is added after the colon but that's not required. Then come the values.
Multiple identical headers may be folded into one single line, delimiting the
values with commas, provided that their order is respected. This is commonly
encountered in the "Cookie:" field. A header may span over multiple lines if
the subsequent lines begin with an LWS. In the example in 1.2, lines 4 and 5
define a total of 3 values for the "Accept:" header.

Contrary to a common mis-conception, header names are not case-sensitive, and
their values are not either if they refer to other header names (such as the
"Connection:" header).

The end of the headers is indicated by the first empty line. People often say
that it's a double line feed, which is not exact, even if a double line feed
is one valid form of empty line.

Fortunately, HAProxy takes care of all these complex combinations when indexing
headers, checking values and counting them, so there is no reason to worry
about the way they could be written, but it is important not to accuse an

application of being buggy if it does unusual, valid things.

Important note:
   As suggested by RFC2616, HAProxy normalizes headers by replacing line breaks
   in the middle of headers by LWS in order to join multi-line headers. This
   is necessary for proper analysis and helps less capable HTTP parsers to work
   correctly and not to be fooled by such complex constructs.

1.3. HTTP response

An HTTP response looks very much like an HTTP request. Both are called HTTP
messages. Let's consider this HTTP response :

  Line   Contents
  number
     1   HTTP/1.1 200 OK
     2   Content-length: 350
     3   Content-Type: text/html

As a special case, HTTP supports so called "Informational responses" as status
codes 1xx. These messages are special in that they don't convey any part of the
response, they're just used as sort of a signaling message to ask a client to
continue to post its request for instance. In the case of a status 100 response
the requested information will be carried by the next non-100 response message
following the informational one. This implies that multiple responses may be
sent to a single request, and that this only works when keep-alive is enabled
(1xx messages are HTTP/1.1 only). HAProxy handles these messages and is able to
correctly forward and skip them, and only process the next non-100 response. As
such, these messages are neither logged nor transformed, unless explicitly
state otherwise. Status 101 messages indicate that the protocol is changing
over the same connection and that haproxy must switch to tunnel mode, just as
if a CONNECT had occurred. Then the Upgrade header would contain additional
information about the type of protocol the connection is switching to.

1.3.1. The Response line

Line 1 is the "response line". It is always composed of 3 fields :

  - a version tag : HTTP/1.1
  - a status code : 200
  - a reason   : OK

The status code is always 3-digit. The first digit indicates a general status :
 - 1xx = informational message to be skipped (eg: 100, 101)
 - 2xx = OK, content is following   (eg: 200, 206)
 - 3xx = OK, no content following   (eg: 302, 304)
 - 4xx = error caused by the client (eg: 401, 403, 404)
 - 5xx = error caused by the server (eg: 500, 502, 503)

Please refer to RFC2616 for the detailed meaning of all such codes. The
"reason" field is just a hint, but is not parsed by clients. Anything can be
found there, but it's a common practice to respect the well-established
messages. It can be composed of one or multiple words, such as "OK", "Found",
or "Authentication Required".

Haproxy may emit the following status codes by itself :

  Code When / reason
   200 access to stats page, and when replying to monitoring requests
   301 when performing a redirection, depending on the configured code
   302 when performing a redirection, depending on the configured code

   303 when performing a redirection, depending on the configured code
   307 when performing a redirection, depending on the configured code
   308 when performing a redirection, depending on the configured code
   400 for an invalid or too large request
   401 when an authentication is required to perform the action (when
        accessing the stats page)
   403 when a request is forbidden by a "block" ACL or "reqdeny" filter
   408 when the request timeout strikes before the request is complete
   500 when haproxy encounters an unrecoverable internal error, such as a
        memory allocation failure, which should never happen
   502 when the server returns an empty, invalid or incomplete response, or
        when an "rspdeny" filter blocks the response.
   503 when no server was available to handle the request, or in response to
        monitoring requests which match the "monitor fail" condition
   504 when the response timeout strikes before the server responds

The error 4xx and 5xx codes above may be customized (see "errorloc" in section
4.2).

1.3.2. The response headers

Response headers work exactly like request headers, and as such, HAProxy uses
the same parsing function for both. Please refer to paragraph 1.2.2 for more
details.

2. Configuring HAProxy

2.1. Configuration file format

HAProxy's configuration process involves 3 major sources of parameters :

  - the arguments from the command-line, which always take precedence
  - the "global" section, which sets process-wide parameters
  - the proxies sections which can take form of "defaults", "listen",
    "frontend" and "backend".

The configuration file syntax consists in lines beginning with a keyword
referenced in this manual, optionally followed by one or several parameters
delimited by spaces. If spaces have to be entered in strings, then they must be
preceded by a backslash ('\') to be escaped. Backslashes also have to be
escaped by doubling them.

2.2. Time format

Some parameters involve values representing time, such as timeouts. These
values are generally expressed in milliseconds (unless explicitly stated
otherwise) but may be expressed in any other unit by suffixing the unit to the
numeric value. It is important to consider this because it will not be repeated
for every keyword. Supported units are :

  - us : microseconds. 1 microsecond = 1/1000000 second
  - ms : milliseconds. 1 millisecond = 1/1000 second. This is the default.
  - s : seconds. 1s = 1000ms
  - m : minutes. 1m = 60s = 60000ms
  - h : hours.   1h = 60m = 3600s = 3600000ms
  - d : days.   1d = 24h = 1440m = 86400s = 86400000ms

```
2.3. Examples
-------------

    # Simple configuration for an HTTP proxy listening on port 80 on all
    # interfaces and forwarding requests to a single backend "servers" with a
    # single server "server1" listening on 127.0.0.1:8000
    global
        daemon
        maxconn 256

    defaults
        mode http
        timeout connect 5000ms
        timeout client 50000ms
        timeout server 50000ms

    frontend http-in
        bind *:80
        default_backend servers

    backend servers
        server server1 127.0.0.1:8000 maxconn 32


    # The same configuration defined with a single listen block. Shorter but
    # less expressive, especially in HTTP mode.
    global
        daemon
        maxconn 256

    defaults
        mode http
        timeout connect 5000ms
        timeout client 50000ms
        timeout server 50000ms

    listen http-in
        bind *:80
        server server1 127.0.0.1:8000 maxconn 32


Assuming haproxy is in $PATH, test these configurations in a shell with:

    $ sudo haproxy -f configuration.conf -c


3. Global parameters
--------------------

Parameters in the "global" section are process-wide and often OS-specific. They
are generally set once for all and do not need being changed once correct. Some
of them have command-line equivalents.

The following keywords are supported in the "global" section :

 * Process management and security
   - ca-base
   - chroot
   - crt-base
   - daemon
   - gid
   - group
   - log
   - log-send-hostname
   - nbproc
```



```
   - pidfile
   - uid
   - ulimit-n
   - user
   - stats
   - ssl-server-verify
   - node
   - description
   - unix-bind

 * Performance tuning
   - max-spread-checks
   - maxconn
   - maxconnrate
   - maxcomprate
   - maxcompcpuusage
   - maxpipes
   - maxsessrate
   - maxsslconn
   - maxsslrate
   - noepoll
   - nokqueue
   - nopoll
   - nosplice
   - nogetaddrinfo
   - spread-checks
   - tune.bufsize
   - tune.chksize
   - tune.comp.maxlevel
   - tune.http.cookielen
   - tune.http.maxhdr
   - tune.idletimer
   - tune.maxaccept
   - tune.maxpollevents
   - tune.maxrewrite
   - tune.pipesize
   - tune.rcvbuf.client
   - tune.rcvbuf.server
   - tune.sndbuf.client
   - tune.sndbuf.server
   - tune.ssl.cachesize
   - tune.ssl.lifetime
   - tune.ssl.force-private-cache
   - tune.ssl.maxrecord
   - tune.ssl.default-dh-param
   - tune.zlib.memlevel
   - tune.zlib.windowsize

 * Debugging
   - debug
   - quiet


3.1. Process management and security
------------------------------------

ca-base <dir>
  Assigns a default directory to fetch SSL CA certificates and CRLs from when a
  relative path is used with "ca-file" or "crl-file" directives. Absolute
  locations specified in "ca-file" and "crl-file" prevail and ignore "ca-base".

chroot <jail dir>
  Changes current directory to <jail dir> and performs a chroot() there before
  dropping privileges. This increases the security level in case an unknown
  vulnerability would be exploited, since it would make it very hard for the
```

```
   attacker to exploit the system. This only works when the process is started
   with superuser privileges. It is important to ensure that <jail_dir> is both
   empty and unwritable to anyone.

cpu-map <"all"|"odd"|"even"|process_num> <cpu-set>...
   On Linux 2.6 and above, it is possible to bind a process to a specific CPU
   set. This means that the process will never run on other CPUs. The "cpu-map"
   directive specifies CPU sets for process sets. The first argument is the
   process number to bind. This process must have a number between 1 and 32 or
   64, depending on the machine's word size, and any process IDs above nbproc
   are ignored. It is possible to specify all processes at once using "all",
   only odd numbers using "odd" or even numbers using "even", just like with the
   "bind-process" directive. The second and forthcoming arguments are CPU sets.
   Each CPU set is either a unique number between 0 and 31 or 63 or a range with
   two such numbers delimited by a dash ('-'). Multiple CPU numbers or ranges
   may be specified, and the processes will be allowed to bind to all of them.
   Obviously, multiple "cpu-map" directives may be specified. Each "cpu-map"
   directive will replace the previous ones when they overlap.

crt-base <dir>
   Assigns a default directory to fetch SSL certificates from when a relative
   path is used with "crtfile" directives. Absolute locations specified after
   "crtfile" prevail and ignore "crt-base".

daemon
   Makes the process fork into background. This is the recommended mode of
   operation. It is equivalent to the command line "-D" argument. It can be
   disabled by the command line "-db" argument.

gid <number>
   Changes the process' group ID to <number>. It is recommended that the group
   ID is dedicated to HAProxy or to a small set of similar daemons. HAProxy must
   be started with a user belonging to this group, or with superuser privileges.
   Note that if haproxy is started from a user having supplementary groups, it
   will only be able to drop these groups if started with superuser privileges.
   See also "group" and "uid".

group <group name>
   Similar to "gid" but uses the GID of group name <group name> from /etc/group.
   See also "gid" and "user".

log <address> [len <length>] <facility> [max level [min level]]
   Adds a global syslog server. Up to two global servers can be defined. They
   will receive logs for startups and exits, as well as all logs from proxies
   configured with "log global".

   <address> can be one of:

         - An IPv4 address optionally followed by a colon and a UDP port. If
           no port is specified, 514 is used by default (the standard syslog
           port).

         - An IPv6 address followed by a colon and optionally a UDP port. If
           no port is specified, 514 is used by default (the standard syslog
           port).

         - A filesystem path to a UNIX domain socket, keeping in mind
           considerations for chroot (be sure the path is accessible inside
           the chroot) and uid/gid (be sure the path is appropriately
           writeable).

         Any part of the address string may reference any number of environment
         variables by preceding their name with a dollar sign ('$') and
         optionally enclosing them with braces ('{}'), similarly to what is done
         in Bourne shell.
```



```

   <length> is an optional maximum line length. Log lines larger than this value
            will be truncated before being sent. The reason is that syslog
            servers act differently on log line length. All servers support the
            default value of 1024, but some servers simply drop larger lines
            while others do log them. If a server supports long lines, it may
            make sense to set this value here in order to avoid truncating long
            lines. Similarly, if a server drops long lines, it is preferable to
            truncate them before sending them. Accepted values are 80 to 65535
            inclusive. The default value of 1024 is generally fine for all
            standard usages. Some specific cases of long captures or
            JSON-formated logs may require larger values.

   <facility> must be one of the 24 standard syslog facilities :

           kern   user   mail   daemon auth   syslog lpr    news
           uucp   cron   auth2  ftp    ntp    audit  alert  cron2
           local0 local1 local2 local3 local4 local5 local6 local7

   An optional level can be specified to filter outgoing messages. By default,
   all messages are sent. If a maximum level is specified, only messages with a
   severity at least as important as this level will be sent. An optional minimum
   level can be specified. If it is set, logs emitted with a more severe level
   than this one will be capped to this level. This is used to avoid sending
   "emerg" messages on all terminals on some default syslog configurations.
   Eight levels are known :

           emerg  alert  crit   err    warning notice info  debug

log-send-hostname [<string>]
   Sets the hostname field in the syslog header. If optional "string" parameter
   is set the header is set to the string contents, otherwise uses the hostname
   of the system. Generally used if one is not relaying logs through an
   intermediate syslog server or for simply customizing the hostname printed in
   the logs.

log-tag <string>
   Sets the tag field in the syslog header to this string. It defaults to the
   program name as launched from the command line, which usually is "haproxy".
   Sometimes it can be useful to differentiate between multiple processes
   running on the same host.

nbproc <number>
   Creates <number> processes when going daemon. This requires the "daemon"
   mode. By default, only one process is created, which is the recommended mode
   of operation. For systems limited to small sets of file descriptors per
   process, it may be needed to fork multiple daemons. USING MULTIPLE PROCESSES
   IS HARDER TO DEBUG AND IS REALLY DISCOURAGED. See also "daemon".

pidfile <pidfile>
   Writes pids of all daemons into file <pidfile>. This option is equivalent to
   the "-p" command line argument. The file must be accessible to the user
   starting the process. See also "daemon".

stats bind-process [ all | odd | even | <number 1-64>[-<number 1-64>] ] ...
   Limits the stats socket to a certain set of processes numbers. By default the
   stats socket is bound to all processes, causing a warning to be emitted when
   nbproc is greater than 1 because there is no way to select the target process
   when connecting. However, by using this setting, it becomes possible to pin
   the stats socket to a specific set of processes, typically the first one. The
   warning will automatically be disabled when this setting is used, whatever
   the number of processes used. The maximum process ID depends on the machine's
   word size (32 or 64). A better option consists in using the "process" setting
   of the "stats socket" line to force the process on each line.

```

```
ssl-default-bind-ciphers <ciphers>
  This setting is only available when support for OpenSSL was built in. It sets
  the default string describing the list of cipher algorithms ("cipher suite")
  that are negotiated during the SSL/TLS handshake for all "bind" lines which
  do not explicitly define theirs. The format of the string is defined in
  "man 1 ciphers" from OpenSSL man pages, and can be for instance a string such
  as "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH" (without quotes). Please check the
  "bind" keyword for more information.

ssl-default-bind-options [<option>]...
  This setting is only available when support for OpenSSL was built in. It sets
  default ssl-options to force on all "bind" lines. Please check the "bind"
  keyword to see available options.

  Example:
        global
           ssl-default-bind-options no-sslv3 no-tls-tickets

ssl-default-server-ciphers <ciphers>
  This setting is only available when support for OpenSSL was built in. It
  sets the default string describing the list of cipher algorithms that are
  negotiated during the SSL/TLS handshake with the server, for all "server"
  lines which do not explicitly define theirs. The format of the string is
  defined in "man 1 ciphers". Please check the "server" keyword for more
  information.

ssl-default-server-options [<option>]...
  This setting is only available when support for OpenSSL was built in. It sets
  default ssl-options to force on all "server" lines. Please check the "server"
  keyword to see available options.

ssl-server-verify [none|required]
  The default behavior for SSL verify on servers side. If specified to 'none',
  servers certificates are not verified. The default is 'required' except if
  forced using cmdline option '-dV'.

stats socket [<address:port>|<path>] [param*]
  Binds a UNIX socket to <path> or a TCPv4/v6 address to <address:port>.
  Connections to this socket will return various statistics outputs and even
  allow some commands to be issued to change some runtime settings. Please
  consult section 9.2 "Unix Socket commands" for more details.

  All parameters supported by "bind" lines are supported, for instance to
  restrict access to some users or their access rights. Please consult
  section 5.1 for more information.

stats timeout <timeout, in milliseconds>
  The default timeout on the stats socket is set to 10 seconds. It is possible
  to change this value with "stats timeout". The value must be passed in
  milliseconds, or be suffixed by a time unit among { us, ms, s, m, h, d }.

stats maxconn <connections>
  By default, the stats socket is limited to 10 concurrent connections. It is
  possible to change this value with "stats maxconn".

uid <number>
  Changes the process' user ID to <number>. It is recommended that the user ID
  is dedicated to HAProxy or to a small set of similar daemons. HAProxy must
  be started with superuser privileges in order to be able to switch to another
  one. See also "gid" and "user".

ulimit-n <number>
  Sets the maximum number of per-process file-descriptors to <number>. By
  default, it is automatically computed, so it is recommended not to use this
  option.
```



```

unix-bind [ prefix <prefix> ] [ mode <mode> ] [ user <user> ] [ uid <uid> ]
          [ group <group> ] [ gid <gid> ]

  Fixes common settings to UNIX listening sockets declared in "bind" statements.
  This is mainly used to simplify declaration of those UNIX sockets and reduce
  the risk of errors, since those settings are most commonly required but are
  also process-specific. The <prefix> setting can be used to force all socket
  path to be relative to that directory. This might be needed to access another
  component's chroot. Note that those paths are resolved before haproxy chroots
  itself, so they are absolute. The <mode>, <user>, <uid>, <group> and <gid>
  all have the same meaning as their homonyms used by the "bind" statement. If
  both are specified, the "bind" statement has priority, meaning that the
  "unix-bind" settings may be seen as process-wide default settings.

user <user name>
  Similar to "uid" but uses the UID of user name <user name> from /etc/passwd.
  See also "uid" and "group".

node <name>
  Only letters, digits, hyphen and underscore are allowed, like in DNS names.

  This statement is useful in HA configurations where two or more processes or
  servers share the same IP address. By setting a different node-name on all
  nodes, it becomes easy to immediately spot what server is handling the
  traffic.

description <text>
  Add a text that describes the instance.

  Please note that it is required to escape certain characters (# for example)
  and this text is inserted into a html page so you should avoid using
  "<" and ">" characters.


3.2. Performance tuning
-----------------------

max-spread-checks <delay in milliseconds>
  By default, haproxy tries to spread the start of health checks across the
  smallest health check interval of all the servers in a farm. The principle is
  to avoid hammering services running on the same server. But when using large
  check intervals (10 seconds or more), the last servers in the farm take some
  time before starting to be tested, which can be a problem. This parameter is
  used to enforce an upper bound on delay between the first and the last check,
  even if the servers' check intervals are larger. When servers run with
  shorter intervals, their intervals will be respected though.

maxconn <number>
  Sets the maximum per-process number of concurrent connections to <number>. It
  is equivalent to the command-line argument "-n". Proxies will stop accepting
  connections when this limit is reached. The "ulimit-n" parameter is
  automatically adjusted according to this value. See also "ulimit-n". Note:
  the "select" poller cannot reliably use more than 1024 file descriptors on
  some platforms. If your platform only supports select and reports "select
  FAILED" on startup, you need to reduce maxconn until it works (slightly
  below 500 in general).

maxconnrate <number>
  Sets the maximum per-process number of connections per second to <number>.
  Proxies will stop accepting connections when this limit is reached. It can be
  used to limit the global capacity regardless of each frontend capacity. It is
  important to note that this can only be used as a service protection measure,
  as there will not necessarily be a fair share between frontends when the
  limit is reached, so it's a good idea to also limit each frontend to some
```

```
   value close to its expected share. Also, lowering tune.maxaccept can improve
   fairness.

maxcomprate <number>
   Sets the maximum per-process input compression rate to <number> kilobytes
   per second.  For each session, if the maximum is reached, the compression
   level will be decreased during the session. If the maximum is reached at the
   beginning of a session, the session will not compress at all. If the maximum
   is not reached, the compression level will be increased up to
   tune.comp.maxlevel.  A value of zero means there is no limit, this is the
   default value.

maxcompcpuusage <number>
   Sets the maximum CPU usage HAProxy can reach before stopping the compression
   for new requests or decreasing the compression level of current requests.
   It works like 'maxcomprate' but measures CPU usage instead of incoming data
   bandwidth. The value is expressed in percent of the CPU used by haproxy. In
   case of multiple processes (nbproc > 1), each process manages its individual
   usage. A value of 100 disable the limit. The default value is 100. Setting
   a lower value will prevent the compression work from slowing the whole
   process down and from introducing high latencies.

maxpipes <number>
   Sets the maximum per-process number of pipes to <number>. Currently, pipes
   are only used by kernel-based tcp splicing. Since a pipe contains two file
   descriptors, the "ulimit-n" value will be increased accordingly. The default
   value is maxconn/4, which seems to be more than enough for most heavy usages.
   The splice code dynamically allocates and releases pipes, and can fall back
   to standard copy, so setting this value too low may only impact performance.

maxsessrate <number>
   Sets the maximum per-process number of sessions per second to <number>.
   Proxies will stop accepting connections when this limit is reached. It can be
   used to limit the global capacity regardless of each frontend capacity. It is
   important to note that this can only be used as a service protection measure,
   as there will not necessarily be a fair share between frontends when the
   limit is reached, so it's a good idea to also limit each frontend to some
   value close to its expected share. Also, lowering tune.maxaccept can improve
   fairness.

maxsslconn <number>
   Sets the maximum per-process number of concurrent SSL connections to
   <number>. By default there is no SSL-specific limit, which means that the
   global maxconn setting will apply to all connections. Setting this limit
   avoids having openssl use too much memory and crash when malloc returns NULL
   (since it unfortunately does not reliably check for such conditions). Note
   that the limit applies both to incoming and outgoing connections, so one
   connection which is deciphered then ciphered accounts for 2 SSL connections.

maxsslrate <number>
   Sets the maximum per-process number of SSL sessions per second to <number>.
   SSL listeners will stop accepting connections when this limit is reached. It
   can be used to limit the global SSL CPU usage regardless of each frontend
   capacity. It is important to note that this can only be used as a service
   protection measure, as there will not necessarily be a fair share between
   frontends when the limit is reached, so it's a good idea to also limit each
   frontend to some value close to its expected share. It is also important to
   note that the sessions are accounted before they enter the SSL stack and not
   after, which also protects the stack against bad handshakes. Also, lowering
   tune.maxaccept can improve fairness.

maxzlibmem <number>
   Sets the maximum amount of RAM in megabytes per process usable by the zlib.
   When the maximum amount is reached, future sessions will not compress as long
   as RAM is unavailable. When sets to 0, there is no limit.
```



```
   The default value is 0. The value is available in bytes on the UNIX socket
   with "show info" on the line "MaxZlibMemUsage", the memory used by zlib is
   "ZlibMemUsage" in bytes.

noepoll
   Disables the use of the "epoll" event polling system on Linux. It is
   equivalent to the command-line argument "-de". The next polling system
   used will generally be "poll". See also "nopoll".

nokqueue
   Disables the use of the "kqueue" event polling system on BSD. It is
   equivalent to the command-line argument "-dk". The next polling system
   used will generally be "poll". See also "nopoll".

nopoll
   Disables the use of the "poll" event polling system. It is equivalent to the
   command-line argument "-dp". The next polling system used will be "select".
   It should never be needed to disable "poll" since it's available on all
   platforms supported by HAProxy. See also "nokqueue" and "noepoll".

nosplice
   Disables the use of kernel tcp splicing between sockets on Linux. It is
   equivalent to the command line argument "-dS".  Data will then be copied
   using conventional and more portable recv/send calls. Kernel tcp splicing is
   limited to some very recent instances of kernel 2.6. Most versions between
   2.6.25 and 2.6.28 are buggy and will forward corrupted data, so they must not
   be used. This option makes it easier to globally disable kernel splicing in
   case of doubt. See also "option splice-auto", "option splice-request" and
   "option splice-response".

nogetaddrinfo
   Disables the use of getaddrinfo(3) for name resolving. It is equivalent to
   the command line argument "-dG". Deprecated gethostbyname(3) will be used.

spread-checks <0..50, in percent>
   Sometimes it is desirable to avoid sending agent and health checks to
   servers at exact intervals, for instance when many logical servers are
   located on the same physical server. With the help of this parameter, it
   becomes possible to add some randomness in the check interval between 0
   and +/- 50%. A value between 2 and 5 seems to show good results. The
   default value remains at 0.

tune.bufsize <number>
   Sets the buffer size to this size (in bytes). Lower values allow more
   sessions to coexist in the same amount of RAM, and higher values allow some
   applications with very large cookies to work. The default value is 16384 and
   can be changed at build time. It is strongly recommended not to change this
   from the default value, as very low values will break some services such as
   statistics, and values larger than default size will increase memory usage,
   possibly causing the system to run out of memory. At least the global maxconn
   parameter should be decreased by the same factor as this one is increased.
   If HTTP request is larger than (tune.bufsize - tune.maxrewrite), haproxy will
   return HTTP 400 (Bad Request) error. Similarly if an HTTP response is larger
   than this size, haproxy will return HTTP 502 (Bad Gateway).

tune.chksize <number>
   Sets the check buffer size to this size (in bytes). Higher values may help
   find string or regex patterns in very large pages, though doing so may imply
   more memory and CPU usage. The default value is 16384 and can be changed at
   build time. It is not recommended to change this value, but to use better
   checks whenever possible.

tune.comp.maxlevel <number>
   Sets the maximum compression level. The compression level affects CPU
   usage during compression. This value affects CPU usage during compression.
```

```
   Each session using compression initializes the compression algorithm with
   this value. The default value is 1.

tune.http.cookielen <number>
   Sets the maximum length of captured cookies. This is the maximum value that
   the "capture cookie xxx len yyy" will be allowed to take, and any upper value
   will automatically be truncated to this one. It is important not to set too
   high a value because all cookie captures still allocate this size whatever
   their configured value (they share a same pool). This value is per request
   per response, so the memory allocated is twice this value per connection.
   When not specified, the limit is set to 63 characters. It is recommended not
   to change this value.

tune.http.maxhdr <number>
   Sets the maximum number of headers in a request. When a request comes with a
   number of headers greater than this value (including the first line), it is
   rejected with a "400 Bad Request" status code. Similarly, too large responses
   are blocked with "502 Bad Gateway". The default value is 101, which is enough
   for all usages, considering that the widely deployed Apache server uses the
   same limit. It can be useful to push this limit further to temporarily allow
   a buggy application to work by the time it gets fixed. Keep in mind that each
   new header consumes 32bits of memory for each session, so don't push this
   limit too high.

tune.idletimer <timeout>
   Sets the duration after which haproxy will consider that an empty buffer is
   probably associated with an idle stream. This is used to optimally adjust
   some packet sizes while forwarding large and small data alternatively. The
   decision to use splice() or to send large buffers in SSL is modulated by this
   parameter. The value is in milliseconds between 0 and 65535. A value of zero
   means that haproxy will not try to detect idle streams. The default is 1000,
   which seems to correctly detect end user pauses (eg: read a page before
   clicking). There should be not reason for changing this value. Please check
   tune.ssl.maxrecord below.

tune.maxaccept <number>
   Sets the maximum number of consecutive connections a process may accept in a
   row before switching to other work. In single process mode, higher numbers
   give better performance at high connection rates. However in multi-process
   modes, keeping a bit of fairness between processes generally is better to
   increase performance. This value applies individually to each listener, so
   that the number of processes a listener is bound to is taken into account.
   This value defaults to 64. In multi-process mode, it is divided by twice
   the number of processes the listener is bound to. Setting this value to -1
   completely disables the limitation. It should normally not be needed to tweak
   this value.

tune.maxpollevents <number>
   Sets the maximum amount of events that can be processed at once in a call to
   the polling system. The default value is adapted to the operating system. It
   has been noticed that reducing it below 200 tends to slightly decrease
   latency at the expense of network bandwidth, and increasing it above 200
   tends to trade latency for slightly increased bandwidth.

tune.maxrewrite <number>
   Sets the reserved buffer space to this size in bytes. The reserved space is
   used for header rewriting or appending. The first reads on sockets will never
   fill more than bufsize-maxrewrite. Historically it has defaulted to half of
   bufsize, though that does not make much sense since there are rarely large
   numbers of headers to add. Setting it too high prevents processing of large
   requests or responses. Setting it too low prevents addition of new headers
   to already large requests or to POST requests. It is generally wise to set it
   to about 1024. It is automatically readjusted to half of bufsize if it is
   larger than that. This means you don't have to worry about it when changing
   bufsize.
```



```

tune.pipesize <number>
   Sets the kernel pipe buffer size to this size (in bytes). By default, pipes
   are the default size for the system. But sometimes when using TCP splicing,
   it can improve performance to increase pipe sizes, especially if it is
   suspected that pipes are not filled and that many calls to splice() are
   performed. This has an impact on the kernel's memory footprint, so this must
   not be changed if impacts are not understood.

tune.rcvbuf.client <number>
tune.rcvbuf.server <number>
   Forces the kernel socket receive buffer size on the client or the server side
   to the specified value in bytes. This value applies to all TCP/HTTP frontends
   and backends. It should normally never be set, and the default size (0) lets
   the kernel autotune this value depending on the amount of available memory.
   However it can sometimes help to set it to very low values (eg: 4096) in
   order to save kernel memory by preventing it from buffering too large amounts
   of received data. Lower values will significantly increase CPU usage though.

tune.sndbuf.client <number>
tune.sndbuf.server <number>
   Forces the kernel socket send buffer size on the client or the server side to
   the specified value in bytes. This value applies to all TCP/HTTP frontends
   and backends. It should normally never be set, and the default size (0) lets
   the kernel autotune this value depending on the amount of available memory.
   However it can sometimes help to set it to very low values (eg: 4096) in
   order to save kernel memory by preventing it from buffering too large amounts
   of received data. Lower values will significantly increase CPU usage though.
   Another use case is to prevent write timeouts with extremely slow clients due
   to the kernel waiting for a large part of the buffer to be read before
   notifying haproxy again.

tune.ssl.cachesize <number>
   Sets the size of the global SSL session cache, in a number of blocks. A block
   is large enough to contain an encoded session without peer certificate.
   An encoded session with peer certificate is stored in multiple blocks
   depending on the size of the peer certificate. A block uses approximately
   200 bytes of memory. The default value may be forced at build time, otherwise
   defaults to 20000.  When the cache is full, the most idle entries are purged
   and reassigned. Higher values reduce the occurrence of such a purge, hence
   the number of CPU-intensive SSL handshakes by ensuring that all users keep
   their session as long as possible. All entries are pre-allocated upon startup
   and are shared between all processes if "nbproc" is greater than 1. Setting
   this value to 0 disables the SSL session cache.

tune.ssl.force-private-cache
   This boolean disables SSL session cache sharing between all processes. It
   should normally not be used since it will force many renegotiations due to
   clients hitting a random process. But it may be required on some operating
   systems where none of the SSL cache synchronization method may be used. In
   this case, adding a first layer of hash-based load balancing before the SSL
   layer might limit the impact of the lack of session sharing.

tune.ssl.lifetime <timeout>
   Sets how long a cached SSL session may remain valid. This time is expressed
   in seconds and defaults to 300 (5 min). It is important to understand that it
   does not guarantee that sessions will last that long, because if the cache is
   full, the longest idle sessions will be purged despite their configured
   lifetime. The real usefulness of this setting is to prevent sessions from
   being used for too long.

tune.ssl.maxrecord <number>
   Sets the maximum amount of bytes passed to SSL_write() at a time. Default
   value 0 means there is no limit. Over SSL/TLS, the client can decipher the
   data only once it has received a full record. With large records, it means
```

```
   that clients might have to download up to 16kB of data before starting to
   process them. Limiting the value can improve page load times on browsers
   located over high latency or low bandwidth networks. It is suggested to find
   optimal values which fit into 1 or 2 TCP segments (generally 1448 bytes over
   Ethernet with TCP timestamps enabled, or 1460 when timestamps are disabled),
   keeping in mind that SSL/TLS add some overhead. Typical values of 1419 and
   2859 gave good results during tests. Use "strace -e trace=write" to find the
   best value. Haproxy will automatically switch to this setting after an idle
   stream has been detected (see tune.idletimer above).

tune.ssl.default-dh-param <number>
   Sets the maximum size of the Diffie-Hellman parameters used for generating
   the ephemeral/temporary Diffie-Hellman key in case of DHE key exchange. The
   final size will try to match the size of the server's RSA (or DSA) key (e.g,
   a 2048 bits temporary DH key for a 2048 bits RSA key), but will not exceed
   this maximum value. Default value if 1024. Only 1024 or higher values are
   allowed. Higher values will increase the CPU load, and values greater than
   1024 bits are not supported by Java 7 and earlier clients. This value is not
   used if static Diffie-Hellman parameters are supplied via the certificate file.

tune.zlib.memlevel <number>
   Sets the memLevel parameter in zlib initialization for each session. It
   defines how much memory should be allocated for the internal compression
   state. A value of 1 uses minimum memory but is slow and reduces compression
   ratio, a value of 9 uses maximum memory for optimal speed.  Can be a value
   between 1 and 9. The default value is 8.

tune.zlib.windowsize <number>
   Sets the window size (the size of the history buffer) as a parameter of the
   zlib initialization for each session. Larger values of this parameter result
   in better compression at the expense of memory usage.  Can be a value between
   8 and 15.  The default value is 15.

3.3. Debugging
--------------

debug
   Enables debug mode which dumps to stdout all exchanges, and disables forking
   into background. It is the equivalent of the command-line argument "-d". It
   should never be used in a production configuration since it may prevent full
   system startup.

quiet
   Do not display any message during startup. It is equivalent to the command-
   line argument "-q".


3.4. Userlists
--------------
It is possible to control access to frontend/backend/listen sections or to
http stats by allowing only authenticated and authorized users. To do this,
it is required to create at least one userlist and to define users.

userlist <listname>
   Creates new userlist with name <listname>. Many independent userlists can be
   used to store authentication & authorization data for independent customers.

group <groupname> [users <user>,<user>,(...)]
   Adds group <groupname> to the current userlist. It is also possible to
   attach users to this group by using a comma separated list of names
   proceeded by "users" keyword.

user <username> [password|insecure-password <password>]
                [groups <group>,<group>,(...)]
   Adds user <username> to the current userlist. Both secure (encrypted) and
```



```
   insecure (unencrypted) passwords can be used. Encrypted passwords are
   evaluated using the crypt(3) function so depending of the system's
   capabilities, different algorithms are supported. For example modern Glibc
   based Linux system supports MD5, SHA-256, SHA-512 and of course classic,
   DES-based method of encrypting passwords.


   Example:
         userlist L1
           group G1 users tiger,scott
           group G2 users xdb,scott

           user tiger password $6$k6y3o.eP$JlKBx9za9667qe4(...)xHSwRv6J.C0/D7cV91
           user scott insecure-password elgato
           user xdb insecure-password hello

         userlist L2
           group G1
           group G2

           user tiger password $6$k6y3o.eP$JlKBx(...)xHSwRv6J.C0/D7cV91 groups G1
           user scott insecure-password elgato groups G1,G2
           user xdb insecure-password hello groups G2

   Please note that both lists are functionally identical.


3.5. Peers
----------
It is possible to synchronize server entries in stick tables between several
haproxy instances over TCP connections in a multi-master fashion. Each instance
pushes its local updates and insertions to remote peers. Server IDs are used to
identify servers remotely, so it is important that configurations look similar
or at least that the same IDs are forced on each server on all participants.
Interrupted exchanges are automatically detected and recovered from the last
known point. In addition, during a soft restart, the old process connects to
the new one using such a TCP connection to push all its entries before the new
process tries to connect to other peers. That ensures very fast replication
during a reload, it typically takes a fraction of a second even for large
tables.

peers <peersect>
   Creates a new peer list with name <peersect>. It is an independent section,
   which is referenced by one or more stick-tables.

disabled
   Disables a peers section. It disables both listening and any synchronization
   related to this section. This is provided to disable synchronization of stick
   tables without having to comment out all "peers" references.

enable
   This re-enables a disabled peers section which was previously disabled.

peer <peername> <ip>:<port>
   Defines a peer inside a peers section.
   If <peername> is set to the local peer name (by default hostname, or forced
   using "-L" command line option), haproxy will listen for incoming remote peer
   connection on <ip>:<port>. Otherwise, <ip>:<port> defines where to connect to
   to join the remote peer, and <peername> is used at the protocol level to
   identify and validate the remote peer on the server side.

   During a soft restart, local peer <ip>:<port> is used by the old instance to
   connect the new one and initiate a complete replication (teaching process).

   It is strongly recommended to have the exact same peers declaration on all
```

```
   peers and to only rely on the "-L" command line argument to change the local
   peer name. This makes it easier to maintain coherent configuration files
   across all peers.

   Any part of the address string may reference any number of environment
   variables by preceding their name with a dollar sign ('$') and optionally
   enclosing them with braces ('{}'), similarly to what is done in Bourne shell.

   Example:
     peers mypeers
         peer haproxy1 192.168.0.1:1024
         peer haproxy2 192.168.0.2:1024
         peer haproxy3 10.2.0.1:1024

     backend mybackend
         mode tcp
         balance roundrobin
         stick-table type ip size 20k peers mypeers
         stick on src

         server srv1 192.168.0.30:80
         server srv2 192.168.0.31:80


4. Proxies
----------

Proxy configuration can be located in a set of sections :
 - defaults <name>
 - frontend <name>
 - backend  <name>
 - listen   <name>

A "defaults" section sets default parameters for all other sections following
its declaration. Those default parameters are reset by the next "defaults"
section. See below for the list of parameters which can be set in a "defaults"
section. The name is optional but its use is encouraged for better readability.

A "frontend" section describes a set of listening sockets accepting client
connections.

A "backend" section describes a set of servers to which the proxy will connect
to forward incoming connections.

A "listen" section defines a complete proxy with its frontend and backend
parts combined in one section. It is generally useful for TCP-only traffic.

All proxy names must be formed from upper and lower case letters, digits,
'-' (dash), '_' (underscore) , '.' (dot) and ':' (colon). ACL names are
case-sensitive, which means that "www" and "WWW" are two different proxies.

Historically, all proxy names could overlap, it just caused troubles in the
logs. Since the introduction of content switching, it is mandatory that two
proxies with overlapping capabilities (frontend/backend) have different names.
However, it is still permitted that a frontend and a backend share the same
name, as this configuration seems to be commonly encountered.

Right now, two major proxy modes are supported : "tcp", also known as layer 4,
and "http", also known as layer 7. In layer 4 mode, HAProxy simply forwards
bidirectional traffic between two sides. In layer 7 mode, HAProxy analyzes the
protocol, and can interact with it by allowing, blocking, switching, adding,
modifying, or removing arbitrary contents in requests or responses, based on
arbitrary criteria.

In HTTP mode, the processing applied to requests and responses flowing over
```



```
a connection depends in the combination of the frontend's HTTP options and
the backend's. HAProxy supports 5 connection modes :

  - KAL : keep alive ("option http-keep-alive") which is the default mode : all
    requests and responses are processed, and connections remain open but idle
    between responses and new requests.

  - TUN: tunnel ("option http-tunnel") : this was the default mode for versions
    1.0 to 1.5-dev21 : only the first request and response are processed, and
    everything else is forwarded with no analysis at all. This mode should not
    be used as it creates lots of trouble with logging and HTTP processing.

  - PCL: passive close ("option httpclose") : exactly the same as tunnel mode,
    but with "Connection: close" appended in both directions to try to make
    both ends close after the first request/response exchange.

  - SCL: server close ("option http-server-close") : the server-facing
    connection is closed after the end of the response is received, but the
    client-facing connection remains open.

  - FCL: forced close ("option forceclose") : the connection is actively closed
    after the end of the response.

The effective mode that will be applied to a connection passing through a
frontend and a backend can be determined by both proxy modes according to the
following matrix, but in short, the modes are symmetric, keep-alive is the
weakest option and force close is the strongest.

                            Backend mode

               | KAL | TUN | PCL | SCL | FCL
          ----+-----+-----+-----+-----+----
          KAL | KAL | TUN | PCL | SCL | FCL
          ----+-----+-----+-----+-----+----
          TUN | TUN | TUN | PCL | SCL | FCL
 Frontend ----+-----+-----+-----+-----+----
   mode   PCL | PCL | PCL | PCL | FCL | FCL
          ----+-----+-----+-----+-----+----
          SCL | SCL | SCL | FCL | SCL | FCL
          ----+-----+-----+-----+-----+----
          FCL | FCL | FCL | FCL | FCL | FCL


4.1. Proxy keywords matrix
--------------------------

The following list of keywords is supported. Most of them may only be used in a
limited set of section types. Some of them are marked as "deprecated" because
they are inherited from an old syntax which may be confusing or functionally
limited, and there are new recommended keywords to replace them. Keywords
marked with "(*)" can be optionally inverted using the "no" prefix, eg. "no
option contstats". This makes sense when the option has been enabled by default
and must be disabled for a specific instance. Such options may also be prefixed
with "default" in order to restore default settings regardless of what has been
specified in a previous "defaults" section.


 keyword                              defaults   frontend   listen    backend
-------------------------------------+----------+----------+---------+---------
acl                                     -          X          X         X
appsession                              -          -          X         X
backlog                                 X          X          X         -
balance                                 X          -          X         X
bind                                    -          X          X         -
```

```
bind-process                                X          X          X          X
block                                       -          X          X          X
capture cookie                              -          X          X          -
capture request header                      -          X          X          -
capture response header                     -          X          X          -
clitimeout                    (deprecated)  X          X          X          -
compression                                 X          X          X          X
contimeout                    (deprecated)  X          -          X          X
cookie                                      X          -          X          X
default-server                              X          -          X          X
default_backend                             X          X          X          -
description                                 -          X          X          X
disabled                                    X          X          X          X
dispatch                                    -          -          X          X
enabled                                     X          X          X          X
errorfile                                   X          X          X          X
errorloc                                    X          X          X          X
errorloc302                                 X          X          X          X
-- keyword -------------------------- defaults - frontend - listen -- backend -
errorloc303                                 X          X          X          X
force-persist                               -          X          X          X
fullconn                                    X          -          X          X
grace                                       X          X          X          X
hash-type                                   X          -          X          X
http-check disable-on-404                   X          -          X          X
http-check expect                           -          -          X          X
http-check send-state                       X          -          X          X
http-request                                -          X          X          X
http-response                               -          X          X          X
http-send-name-header                       -          -          X          X
id                                          -          X          X          X
ignore-persist                              -          X          X          X
log                                   (*)   X          X          X          X
log-format                                  X          X          X          -
max-keep-alive-queue                        X          -          X          X
maxconn                                     X          X          X          -
mode                                        X          X          X          X
monitor fail                                -          X          X          -
monitor-net                                 X          X          X          -
monitor-uri                                 X          X          X          -
option abortonclose                   (*)   X          -          X          X
option accept-invalid-http-request    (*)   X          X          X          -
option accept-invalid-http-response   (*)   X          -          X          X
option allbackups                     (*)   X          -          X          X
option checkcache                     (*)   X          -          X          X
option clitcpka                       (*)   X          X          X          -
option contstats                      (*)   X          X          X          -
option dontlog-normal                 (*)   X          X          X          -
option dontlognull                    (*)   X          X          X          -
option forceclose                     (*)   X          X          X          X
-- keyword -------------------------- defaults - frontend - listen -- backend -
option forwardfor                           X          X          X          X
option http-ignore-probes             (*)   X          X          X          -
option http-keep-alive                (*)   X          X          X          X
option http-no-delay                  (*)   X          X          X          X
option http-pretend-keepalive         (*)   X          X          X          X
option http-server-close              (*)   X          X          X          X
option http-tunnel                    (*)   X          X          X          X
option http-use-proxy-header          (*)   X          X          X          -
option httpchk                              X          -          X          X
option httpclose                      (*)   X          X          X          X
option httplog                              X          X          X          X
option http_proxy                     (*)   X          X          X          X
option independent-streams            (*)   X          X          X          X
option ldap-check                           X          -          X          X
```



```
option log-health-checks              (*)   X          -          X          X
option log-separate-errors            (*)   X          X          X          -
option logasap                        (*)   X          X          X          -
option mysql-check                          X          -          X          X
option nolinger                       (*)   X          X          X          X
option originalto                           X          X          X          X
option persist                        (*)   X          -          X          X
option pgsql-check                          X          -          X          X
option prefer-last-server             (*)   X          -          X          X
option redispatch                     (*)   X          -          X          X
option redis-check                          X          -          X          X
option smtpchk                              X          -          X          X
option socket-stats                   (*)   X          X          X          -
option splice-auto                    (*)   X          X          X          X
option splice-request                 (*)   X          X          X          X
option splice-response                (*)   X          X          X          X
option srvtcpka                       (*)   X          -          X          X
option ssl-hello-chk                        X          -          X          X
-- keyword -------------------------- defaults - frontend - listen -- backend -
option tcp-check                            X          -          X          X
option tcp-smart-accept               (*)   X          X          X          -
option tcp-smart-connect              (*)   X          -          X          X
option tcpka                                X          X          X          X
option tcplog                               X          X          X          X
option transparent                    (*)   X          -          X          X
persist rdp-cookie                          X          -          X          X
rate-limit sessions                         X          X          X          -
redirect                                    -          X          X          X
redisp                        (deprecated)  X          -          X          X
redispatch                    (deprecated)  X          -          X          X
reqadd                                      -          X          X          X
reqallow                                    -          X          X          X
reqdel                                      -          X          X          X
reqdeny                                     -          X          X          X
reqiallow                                   -          X          X          X
reqidel                                     -          X          X          X
reqideny                                    -          X          X          X
reqipass                                    -          X          X          X
reqirep                                     -          X          X          X
reqisetbe                                   -          X          X          X
reqitarpit                                  -          X          X          X
reqpass                                     -          X          X          X
reqrep                                      -          X          X          X
-- keyword -------------------------- defaults - frontend - listen -- backend -
reqsetbe                                    -          X          X          X
reqtarpit                                   -          X          X          X
retries                                     X          -          X          X
rspadd                                      -          X          X          X
rspdel                                      -          X          X          X
rspdeny                                     -          X          X          X
rspidel                                     -          X          X          X
rspideny                                    -          X          X          X
rspirep                                     -          X          X          X
rsprep                                      -          X          X          X
server                                      -          -          X          X
source                                      X          -          X          X
srvtimeout                    (deprecated)  X          -          X          X
stats admin                                 -          X          X          X
stats auth                                  X          X          X          X
stats enable                                X          X          X          X
stats hide-version                          X          X          X          X
stats http-request                          -          X          X          X
stats realm                                 X          X          X          X
stats refresh                               X          X          X          X
stats scope                                 X          X          X          X
```

```
stats show-desc                           X          X         X         X
stats show-legends                        X          X         X         X
stats show-node                           X          X         X         X
stats uri                                 X          X         X         X
-- keyword -------------------------- defaults - frontend - listen -- backend -
stick match                               -          -         X         X
stick on                                  -          -         X         X
stick store-request                       -          -         X         X
stick store-response                      -          -         X         X
stick-table                               -          -         X         X
tcp-check connect                         -          -         X         X
tcp-check expect                          -          -         X         X
tcp-check send                            -          -         X         X
tcp-check send-binary                     -          -         X         X
tcp-request connection                    -          X         X         -
tcp-request content                       -          X         X         X
tcp-request inspect-delay                 -          X         X         X
tcp-response content                      -          -         X         X
tcp-response inspect-delay                -          -         X         X
timeout check                             X          -         X         X
timeout client                            X          X         X         -
timeout client-fin                        X          X         X         -
timeout clitimeout          (deprecated)  X          X         X         -
timeout connect                           X          -         X         X
timeout contimeout          (deprecated)  X          -         X         X
timeout http-keep-alive                   X          X         X         X
timeout http-request                      X          X         X         X
timeout queue                             X          -         X         X
timeout server                            X          -         X         X
timeout server-fin                        X          -         X         X
timeout srvtimeout          (deprecated)  X          -         X         X
timeout tarpit                            X          X         X         X
timeout tunnel                            X          -         X         X
transparent                 (deprecated)  X          -         X         X
unique-id-format                          X          X         X         -
unique-id-header                          X          X         X         -
use_backend                               -          X         X         -
use-server                                -          -         X         X
-----------------------------------+----------+----------+---------+---------
 keyword                           defaults   frontend   listen    backend
```

4.2. Alphabetically sorted keywords reference

This section provides a description of each keyword and its usage.

```
acl <aclname> <criterion> [flags] [operator] <value> ...
  Declare or complete an access list.
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes
  Example:
        acl invalid_src  src          0.0.0.0/7 224.0.0.0/3
        acl invalid_src  src_port     0:1023
        acl local_dst    hdr(host) -i localhost

  See section 7 about ACL usage.


appsession <cookie> len <length> timeout <holdtime>
           [request-learn] [prefix] [mode <path-parameters|query-string>]
  Define session stickiness on an existing application cookie.
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    no    |   yes  |   yes
```



```
  Arguments :
    <cookie>   this is the name of the cookie used by the application and which
               HAProxy will have to learn for each new session.

    <length>   this is the max number of characters that will be memorized and
               checked in each cookie value.

    <holdtime> this is the time after which the cookie will be removed from
               memory if unused. If no unit is specified, this time is in
               milliseconds.

    request-learn
               If this option is specified, then haproxy will be able to learn
               the cookie found in the request in case the server does not
               specify any in response. This is typically what happens with
               PHPSESSID cookies, or when haproxy's session expires before
               the application's session and the correct server is selected.
               It is recommended to specify this option to improve reliability.

    prefix     When this option is specified, haproxy will match on the cookie
               prefix (or URL parameter prefix). The appsession value is the
               data following this prefix.

               Example :
               appsession ASPSESSIONID len 64 timeout 3h prefix

               This will match the cookie ASPSESSIONIDXXXX=XXXXX,
               the appsession value will be XXXX=XXXXX.

    mode       This option allows to change the URL parser mode.
               2 modes are currently supported :
               - path-parameters :
                 The parser looks for the appsession in the path parameters
                 part (each parameter is separated by a semi-colon), which is
                 convenient for JSESSIONID for example.
                 This is the default mode if the option is not set.
               - query-string :
                 In this mode, the parser will look for the appsession in the
                 query string.

  When an application cookie is defined in a backend, HAProxy will check when
  the server sets such a cookie, and will store its value in a table, and
  associate it with the server's identifier. Up to <length> characters from
  the value will be retained. On each connection, haproxy will look for this
  cookie both in the "Cookie:" headers, and as a URL parameter (depending on
  the mode used). If a known value is found, the client will be directed to the
  server associated with this value. Otherwise, the load balancing algorithm is
  applied. Cookies are automatically removed from memory when they have been
  unused for a duration longer than <holdtime>.

  The definition of an application cookie is limited to one per backend.

  Note : Consider not using this feature in multi-process mode (nbproc > 1)
         unless you know what you do : memory is not shared between the
         processes, which can result in random behaviours.

  Example :
        appsession JSESSIONID len 52 timeout 3h

  See also : "cookie", "capture cookie", "balance", "stick", "stick-table",
             "ignore-persist", "nbproc" and "bind-process".


backlog <conns>
  Give hints to the system about the approximate listen backlog desired size
```

```
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments :
    <conns>   is the number of pending connections. Depending on the operating
              system, it may represent the number of already acknowledged
              connections, of non-acknowledged ones, or both.

  In order to protect against SYN flood attacks, one solution is to increase
  the system's SYN backlog size. Depending on the system, sometimes it is just
  tunable via a system parameter, sometimes it is not adjustable at all, and
  sometimes the system relies on hints given by the application at the time of
  the listen() syscall. By default, HAProxy passes the frontend's maxconn value
  to the listen() syscall. On systems which can make use of this value, it can
  sometimes be useful to be able to specify a different value, hence this
  backlog parameter.

  On Linux 2.4, the parameter is ignored by the system. On Linux 2.6, it is
  used as a hint and the system accepts up to the smallest greater power of
  two, and never more than some limits (usually 32768).

  See also : "maxconn" and the target operating system's tuning guide.


balance <algorithm> [ <arguments> ]
balance url_param <param> [check_post]
  Define the load balancing algorithm to be used in a backend.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <algorithm> is the algorithm used to select a server when doing load
                balancing. This only applies when no persistence information
                is available, or when a connection is redispatched to another
                server. <algorithm> may be one of the following :

      roundrobin  Each server is used in turns, according to their weights.
                  This is the smoothest and fairest algorithm when the server's
                  processing time remains equally distributed. This algorithm
                  is dynamic, which means that server weights may be adjusted
                  on the fly for slow starts for instance. It is limited by
                  design to 4095 active servers per backend. Note that in some
                  large farms, when a server becomes up after having been down
                  for a very short time, it may sometimes take a few hundreds
                  requests for it to be re-integrated into the farm and start
                  receiving traffic. This is normal, though very rare. It is
                  indicated here in case you would have the chance to observe
                  it, so that you don't worry.

      static-rr   Each server is used in turns, according to their weights.
                  This algorithm is as similar to roundrobin except that it is
                  static, which means that changing a server's weight on the
                  fly will have no effect. On the other hand, it has no design
                  limitation on the number of servers, and when a server goes
                  up, it is always immediately reintroduced into the farm, once
                  the full map is recomputed. It also uses slightly less CPU to
                  run (around -1%).

      leastconn   The server with the lowest number of connections receives the
                  connection. Round-robin is performed within groups of servers
                  of the same load to ensure that all servers will be used. Use
                  of this algorithm is recommended where very long sessions are
                  expected, such as LDAP, SQL, TSE, etc... but is not very well
                  suited for protocols using short sessions such as HTTP. This
                  algorithm is dynamic, which means that server weights may be
                  adjusted on the fly for slow starts for instance.
```



```
      first       The first server with available connection slots receives the
                  connection. The servers are chosen from the lowest numeric
                  identifier to the highest (see server parameter "id"), which
                  defaults to the server's position in the farm. Once a server
                  reaches its maxconn value, the next server is used. It does
                  not make sense to use this algorithm without setting maxconn.
                  The purpose of this algorithm is to always use the smallest
                  number of servers so that extra servers can be powered off
                  during non-intensive hours. This algorithm ignores the server
                  weight, and brings more benefit to long session such as RDP
                  or IMAP than HTTP, though it can be useful there too. In
                  order to use this algorithm efficiently, it is recommended
                  that a cloud controller regularly checks server usage to turn
                  them off when unused, and regularly checks backend queue to
                  turn new servers on when the queue inflates. Alternatively,
                  using "http-check send-state" may inform servers on the load.

      source      The source IP address is hashed and divided by the total
                  weight of the running servers to designate which server will
                  receive the request. This ensures that the same client IP
                  address will always reach the same server as long as no
                  server goes down or up. If the hash result changes due to the
                  number of running servers changing, many clients will be
                  directed to a different server. This algorithm is generally
                  used in TCP mode where no cookie may be inserted. It may also
                  be used on the Internet to provide a best-effort stickiness
                  to clients which refuse session cookies. This algorithm is
                  static by default, which means that changing a server's
                  weight on the fly will have no effect, but this can be
                  changed using "hash-type".

      uri         This algorithm hashes either the left part of the URI (before
                  the question mark) or the whole URI (if the "whole" parameter
                  is present) and divides the hash value by the total weight of
                  the running servers. The result designates which server will
                  receive the request. This ensures that the same URI will
                  always be directed to the same server as long as no server
                  goes up or down. This is used with proxy caches and
                  anti-virus proxies in order to maximize the cache hit rate.
                  Note that this algorithm may only be used in an HTTP backend.
                  This algorithm is static by default, which means that
                  changing a server's weight on the fly will have no effect,
                  but this can be changed using "hash-type".

                  This algorithm supports two optional parameters "len" and
                  "depth", both followed by a positive integer number. These
                  options may be helpful when it is needed to balance servers
                  based on the beginning of the URI only. The "len" parameter
                  indicates that the algorithm should only consider that many
                  characters at the beginning of the URI to compute the hash.
                  Note that having "len" set to 1 rarely makes sense since most
                  URIs start with a leading "/".

                  The "depth" parameter indicates the maximum directory depth
                  to be used to compute the hash. One level is counted for each
                  slash in the request. If both parameters are specified, the
                  evaluation stops when either is reached.

      url_param   The URL parameter specified in argument will be looked up in
                  the query string of each HTTP GET request.

                  If the modifier "check_post" is used, then an HTTP POST
                  request entity will be searched for the parameter argument,
                  when it is not found in a query string after a question mark
                  ('?') in the URL. The message body will only start to be
```

```
                    analyzed once either the advertised amount of data has been
                    received or the request buffer is full. In the unlikely event
                    that chunked encoding is used, only the first chunk is
                    scanned. Parameter values separated by a chunk boundary, may
                    be randomly balanced if at all. This keyword used to support
                    an optional <max_wait> parameter which is now ignored.

                    If the parameter is found followed by an equal sign ('=') and
                    a value, then the value is hashed and divided by the total
                    weight of the running servers. The result designates which
                    server will receive the request.

                    This is used to track user identifiers in requests and ensure
                    that a same user ID will always be sent to the same server as
                    long as no server goes up or down. If no value is found or if
                    the parameter is not found, then a round robin algorithm is
                    applied. Note that this algorithm may only be used in an HTTP
                    backend. This algorithm is static by default, which means
                    that changing a server's weight on the fly will have no
                    effect, but this can be changed using "hash-type".

      hdr(<name>) The HTTP header <name> will be looked up in each HTTP
                    request. Just as with the equivalent ACL 'hdr()' function,
                    the header name in parenthesis is not case sensitive. If the
                    header is absent or if it does not contain any value, the
                    roundrobin algorithm is applied instead.

                    An optional 'use_domain_only' parameter is available, for
                    reducing the hash algorithm to the main domain part with some
                    specific headers such as 'Host'. For instance, in the Host
                    value "haproxy.1wt.eu", only "1wt" will be considered.

                    This algorithm is static by default, which means that
                    changing a server's weight on the fly will have no effect,
                    but this can be changed using "hash-type".

      rdp-cookie
      rdp-cookie(<name>)
                    The RDP cookie <name> (or "mstshash" if omitted) will be
                    looked up and hashed for each incoming TCP request. Just as
                    with the equivalent ACL 'req_rdp_cookie()' function, the name
                    is not case-sensitive. This mechanism is useful as a degraded
                    persistence mode, as it makes it possible to always send the
                    same user (or the same session ID) to the same server. If the
                    cookie is not found, the normal roundrobin algorithm is
                    used instead.

                    Note that for this to work, the frontend must ensure that an
                    RDP cookie is already present in the request buffer. For this
                    you must use 'tcp-request content accept' rule combined with
                    a 'req_rdp_cookie_cnt' ACL.

                    This algorithm is static by default, which means that
                    changing a server's weight on the fly will have no effect,
                    but this can be changed using "hash-type".

                    See also the rdp_cookie pattern fetch function.

    <arguments> is an optional list of arguments which may be needed by some
                algorithms. Right now, only "url_param" and "uri" support an
                optional argument.

  The load balancing algorithm of a backend is set to roundrobin when no other
  algorithm, mode nor option have been set. The algorithm may only be set once
  for each backend.
```



```

  Examples :
        balance roundrobin
        balance url_param userid
        balance url_param session_id check_post 64
        balance hdr(User-Agent)
        balance hdr(host)
        balance hdr(Host) use_domain_only

  Note: the following caveats and limitations on using the "check_post"
  extension with "url_param" must be considered :

    - all POST requests are eligible for consideration, because there is no way
      to determine if the parameters will be found in the body or entity which
      may contain binary data. Therefore another method may be required to
      restrict consideration of POST requests that have no URL parameters in
      the body. (see acl reqideny http_end)

    - using a <max_wait> value larger than the request buffer size does not
      make sense and is useless. The buffer size is set at build time, and
      defaults to 16 kB.

    - Content-Encoding is not supported, the parameter search will probably
      fail; and load balancing will fall back to Round Robin.

    - Expect: 100-continue is not supported, load balancing will fall back to
      Round Robin.

    - Transfer-Encoding (RFC2616 3.6.1) is only supported in the first chunk.
      If the entire parameter value is not present in the first chunk, the
      selection of server is undefined (actually, defined by how little
      actually appeared in the first chunk).

    - This feature does not support generation of a 100, 411 or 501 response.

    - In some cases, requesting "check_post" MAY attempt to scan the entire
      contents of a message body. Scanning normally terminates when linear
      white space or control characters are found, indicating the end of what
      might be a URL parameter list. This is probably not a concern with SGML
      type message bodies.

  See also : "dispatch", "cookie", "appsession", "transparent", "hash-type" and
             "http_proxy".


bind [<address>]:<port_range> [, ...] [param*]
bind /<path> [, ...] [param*]
  Define one or several listening addresses and/or ports in a frontend.
  May be used in sections :   defaults | frontend | listen | backend
                                  no   |    yes   |   yes  |   no
  Arguments :
    <address>     is optional and can be a host name, an IPv4 address, an IPv6
                  address, or '*'. It designates the address the frontend will
                  listen on. If unset, all IPv4 addresses of the system will be
                  listened on. The same will apply for '*' or the system's
                  special address "0.0.0.0". The IPv6 equivalent is '::'.
                  Optionally, an address family prefix may be used before the
                  address to force the family regardless of the address format,
                  which can be useful to specify a path to a unix socket with
                  no slash ('/'). Currently supported prefixes are :
                    - 'ipv4@'  -> address is always IPv4
                    - 'ipv6@'  -> address is always IPv6
                    - 'unix@'  -> address is a path to a local unix socket
                    - 'abns@'  -> address is in abstract namespace (Linux only).
                      Note: since abstract sockets are not "rebindable", they
```

```
                              do not cope well with multi-process mode during
                              soft-restart, so it is better to avoid them if
                              nbproc is greater than 1. The effect is that if the
                              new process fails to start, only one of the old ones
                              will be able to rebind to the socket.
                        - 'fd@<n>' -> use file descriptor <n> inherited from the
                          parent. The fd must be bound and may or may not already
                          be listening.
                    Any part of the address string may reference any number of
                    environment variables by preceding their name with a dollar
                    sign ('$') and optionally enclosing them with braces ('{}'),
                    similarly to what is done in Bourne shell.

      <port_range>  is either a unique TCP port, or a port range for which the
                    proxy will accept connections for the IP address specified
                    above. The port is mandatory for TCP listeners. Note that in
                    the case of an IPv6 address, the port is always the number
                    after the last colon (':'). A range can either be :
                     - a numerical port (ex: '80')
                     - a dash-delimited ports range explicitly stating the lower
                       and upper bounds (ex: '2000-2100') which are included in
                       the range.

                    Particular care must be taken against port ranges, because
                    every <address:port> couple consumes one socket (= a file
                    descriptor), so it's easy to consume lots of descriptors
                    with a simple range, and to run out of sockets. Also, each
                    <address:port> couple must be used only once among all
                    instances running on a same system. Please note that binding
                    to ports lower than 1024 generally require particular
                    privileges to start the program, which are independent of
                    the 'uid' parameter.

      <path>        is a UNIX socket path beginning with a slash ('/'). This is
                    alternative to the TCP listening port. Haproxy will then
                    receive UNIX connections on the socket located at this place.
                    The path must begin with a slash and by default is absolute.
                    It can be relative to the prefix defined by "unix-bind" in
                    the global section. Note that the total length of the prefix
                    followed by the socket path cannot exceed some system limits
                    for UNIX sockets, which commonly are set to 107 characters.

      <param*>      is a list of parameters common to all sockets declared on the
                    same line. These numerous parameters depend on OS and build
                    options and have a complete section dedicated to them. Please
                    refer to section 5 to for more details.

    It is possible to specify a list of address:port combinations delimited by
    commas. The frontend will then listen on all of these addresses. There is no
    fixed limit to the number of addresses and ports which can be listened on in
    a frontend, as well as there is no limit to the number of "bind" statements
    in a frontend.

    Example :
          listen http_proxy
              bind :80,:443
              bind 10.0.0.1:10080,10.0.0.1:10443
              bind /var/run/ssl-frontend.sock user root mode 600 accept-proxy

          listen http_https_proxy
              bind :80
              bind :443 ssl crt /etc/haproxy/site.pem

          listen http_https_proxy_explicit
              bind ipv6@:80
```



```
              bind ipv4@public_ssl:443 ssl crt /etc/haproxy/site.pem
              bind unix@ssl-frontend.sock user root mode 600 accept-proxy

          listen external_bind_app1
              bind fd@${FD_APP1}

    Note: regarding Linux's abstract namespace sockets, HAProxy uses the whole
          sun_path length is used for the address length. Some other programs
          such as socat use the string length only by default. Pass the option
          ",unix-tightsocklen=0" to any abstract socket definition in socat to
          make it compatible with HAProxy's.

    See also : "source", "option forwardfor", "unix-bind" and the PROXY protocol
               documentation, and section 5 about bind options.


  bind-process [ all | odd | even | <number 1-64>[-<number 1-64>] ] ...
    Limit visibility of an instance to a certain set of processes numbers.
    May be used in sections :   defaults | frontend | listen | backend
                                    yes   |    yes   |   yes  |   yes
    Arguments :
      all           All process will see this instance. This is the default. It
                    may be used to override a default value.

      odd           This instance will be enabled on processes 1,3,5,...63. This
                    option may be combined with other numbers.

      even          This instance will be enabled on processes 2,4,6,...64. This
                    option may be combined with other numbers. Do not use it
                    with less than 2 processes otherwise some instances might be
                    missing from all processes.

      number        The instance will be enabled on this process number or range,
                    whose values must all be between 1 and 32 or 64 depending on
                    the machine's word size. If a proxy is bound to process
                    numbers greater than the configured global.nbproc, it will
                    either be forced to process #1 if a single process was
                    specified, or to all processes otherwise.

    This keyword limits binding of certain instances to certain processes. This
    is useful in order not to have too many processes listening to the same
    ports. For instance, on a dual-core machine, it might make sense to set
    'nbproc 2' in the global section, then distributes the listeners among 'odd'
    and 'even' instances.

    At the moment, it is not possible to reference more than 32 or 64 processes
    using this keyword, but this should be more than enough for most setups.
    Please note that 'all' really means all processes regardless of the machine's
    word size, and is not limited to the first 32 or 64.

    Each "bind" line may further be limited to a subset of the proxy's processes,
    please consult the "process" bind keyword in section 5.1.

    When a frontend has no explicit "bind-process" line, it tries to bind to all
    the processes referenced by its "bind" lines. That means that frontends can
    easily adapt to their listeners' processes.

    If some backends are referenced by frontends bound to other processes, the
    backend automatically inherits the frontend's processes.

    Example :
          listen app_ip1
              bind 10.0.0.1:80
              bind-process odd
```

```
        listen app_ip2
            bind 10.0.0.2:80
            bind-process even

        listen management
            bind 10.0.0.3:80
            bind-process 1 2 3 4

        listen management
            bind 10.0.0.4:80
            bind-process 1-4

  See also : "nbproc" in global section, and "process" in section 5.1.


block { if | unless } <condition>
  Block a layer 7 request if/unless a condition is matched
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes

  The HTTP request will be blocked very early in the layer 7 processing
  if/unless <condition> is matched. A 403 error will be returned if the request
  is blocked. The condition has to reference ACLs (see section 7). This is
  typically used to deny access to certain sensitive resources if some
  conditions are met or not met. There is no fixed limit to the number of
  "block" statements per instance.

  Example:
        acl invalid_src  src          0.0.0.0/7 224.0.0.0/3
        acl invalid_src  src_port     0:1023
        acl local_dst    hdr(host) -i localhost
        block if invalid_src || local_dst

  See section 7 about ACL usage.


capture cookie <name> len <length>
  Capture and log a cookie in the request and in the response.
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   no
  Arguments :
    <name>    is the beginning of the name of the cookie to capture. In order
              to match the exact name, simply suffix the name with an equal
              sign ('='). The full name will appear in the logs, which is
              useful with application servers which adjust both the cookie name
              and value (eg: ASPSESSIONXXXXX).

    <length>  is the maximum number of characters to report in the logs, which
              include the cookie name, the equal sign and the value, all in the
              standard "name=value" form. The string will be truncated on the
              right if it exceeds <length>.

  Only the first cookie is captured. Both the "cookie" request headers and the
  "set-cookie" response headers are monitored. This is particularly useful to
  check for application bugs causing session crossing or stealing between
  users, because generally the user's cookies can only change on a login page.

  When the cookie was not presented by the client, the associated log column
  will report "-". When a request does not cause a cookie to be assigned by the
  server, a "-" is reported in the response column.

  The capture is performed in the frontend only because it is necessary that
  the log format does not change for a given frontend depending on the
  backends. This may change in the future. Note that there can be only one
  "capture cookie" statement in a frontend. The maximum capture length is set
```



```
  by the global "tune.http.cookielen" setting and defaults to 63 characters. It
  is not possible to specify a capture in a "defaults" section.

  Example:
        capture cookie ASPSESSION len 32

  See also : "capture request header", "capture response header" as well as
             section 8 about logging.


capture request header <name> len <length>
  Capture and log the last occurrence of the specified request header.
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   no
  Arguments :
    <name>    is the name of the header to capture. The header names are not
              case-sensitive, but it is a common practice to write them as they
              appear in the requests, with the first letter of each word in
              upper case. The header name will not appear in the logs, only the
              value is reported, but the position in the logs is respected.

    <length>  is the maximum number of characters to extract from the value and
              report in the logs. The string will be truncated on the right if
              it exceeds <length>.

  The complete value of the last occurrence of the header is captured. The
  value will be added to the logs between braces ('{}'). If multiple headers
  are captured, they will be delimited by a vertical bar ('|') and will appear
  in the same order they were declared in the configuration. Non-existent
  headers will be logged just as an empty string. Common uses for request
  header captures include the "Host" field in virtual hosting environments, the
  "Content-length" when uploads are supported, "User-agent" to quickly
  differentiate between real users and robots, and "X-Forwarded-For" in proxied
  environments to find where the request came from.

  Note that when capturing headers such as "User-agent", some spaces may be
  logged, making the log analysis more difficult. Thus be careful about what
  you log if you know your log parser is not smart enough to rely on the
  braces.

  There is no limit to the number of captured request headers nor to their
  length, though it is wise to keep them low to limit memory usage per session.
  In order to keep log format consistent for a same frontend, header captures
  can only be declared in a frontend. It is not possible to specify a capture
  in a "defaults" section.

  Example:
        capture request header Host len 15
        capture request header X-Forwarded-For len 15
        capture request header Referer len 15

  See also : "capture cookie", "capture response header" as well as section 8
             about logging.


capture response header <name> len <length>
  Capture and log the last occurrence of the specified response header.
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   no
  Arguments :
    <name>    is the name of the header to capture. The header names are not
              case-sensitive, but it is a common practice to write them as they
              appear in the response, with the first letter of each word in
              upper case. The header name will not appear in the logs, only the
              value is reported, but the position in the logs is respected.
```

```
2081
2082      <length>  is the maximum number of characters to extract from the value and
2083                report in the logs. The string will be truncated on the right if
2084                it exceeds <length>.
2085
2086    The complete value of the last occurrence of the header is captured. The
2087    result will be added to the logs between braces ('{}') after the captured
2088    request headers. If multiple headers are captured, they will be delimited by
2089    a vertical bar ('|') and will appear in the same order they were declared in
2090    the configuration. Non-existent headers will be logged just as an empty
2091    string. Common uses for response header captures include the "Content-length"
2092    header which indicates how many bytes are expected to be returned, the
2093    "Location" header to track redirections.
2094
2095    There is no limit to the number of captured response headers nor to their
2096    length, though it is wise to keep them low to limit memory usage per session.
2097    In order to keep log format consistent for a same frontend, header captures
2098    can only be declared in a frontend. It is not possible to specify a capture
2099    in a "defaults" section.
2100
2101    Example:
2102          capture response header Content-length len 9
2103          capture response header Location len 15
2104
2105    See also : "capture cookie", "capture request header" as well as section 8
2106               about logging.
2107
2108
2109  clitimeout <timeout> (deprecated)
2110    Set the maximum inactivity time on the client side.
2111    May be used in sections :   defaults | frontend | listen | backend
2112                                   yes   |    yes   |   yes  |   no
2113    Arguments :
2114      <timeout> is the timeout value is specified in milliseconds by default, but
2115                can be in any other unit if the number is suffixed by the unit,
2116                as explained at the top of this document.
2117
2118    The inactivity timeout applies when the client is expected to acknowledge or
2119    send data. In HTTP mode, this timeout is particularly important to consider
2120    during the first phase, when the client sends the request, and during the
2121    response while it is reading data sent by the server. The value is specified
2122    in milliseconds by default, but can be in any other unit if the number is
2123    suffixed by the unit, as specified at the top of this document. In TCP mode
2124    (and to a lesser extent, in HTTP mode), it is highly recommended that the
2125    client timeout remains equal to the server timeout in order to avoid complex
2126    situations to debug. It is a good practice to cover one or several TCP packet
2127    losses by specifying timeouts that are slightly above multiples of 3 seconds
2128    (eg: 4 or 5 seconds).
2129
2130    This parameter is specific to frontends, but can be specified once for all in
2131    "defaults" sections. This is in fact one of the easiest solutions not to
2132    forget about it. An unspecified timeout results in an infinite timeout, which
2133    is not recommended. Such a usage is accepted and works but reports a warning
2134    during startup because it may results in accumulation of expired sessions in
2135    the system if the system's timeouts are not configured either.
2136
2137    This parameter is provided for compatibility but is currently deprecated.
2138    Please use "timeout client" instead.
2139
2140    See also : "timeout client", "timeout http-request", "timeout server", and
2141               "srvtimeout".
2142
2143  compression algo <algorithm> ...
2144  compression type <mime type> ...
2145  compression offload
```



```
2146    Enable HTTP compression.
2147    May be used in sections :   defaults | frontend | listen | backend
2148                                   yes   |    yes   |   yes  |   yes
2149    Arguments :
2150      algo     is followed by the list of supported compression algorithms.
2151      type     is followed by the list of MIME types that will be compressed.
2152      offload  makes haproxy work as a compression offloader only (see notes).
2153
2154    The currently supported algorithms are :
2155      identity  this is mostly for debugging, and it was useful for developing
2156                the compression feature. Identity does not apply any change on
2157                data.
2158
2159      gzip      applies gzip compression. This setting is only available when
2160                support for zlib was built in.
2161
2162      deflate   same as gzip, but with deflate algorithm and zlib format.
2163                Note that this algorithm has ambiguous support on many browsers
2164                and no support at all from recent ones. It is strongly
2165                recommended not to use it for anything else than experimentation.
2166                This setting is only available when support for zlib was built
2167                in.
2168
2169    Compression will be activated depending on the Accept-Encoding request
2170    header. With identity, it does not take care of that header.
2171    If backend servers support HTTP compression, these directives
2172    will be no-op: haproxy will see the compressed response and will not
2173    compress again. If backend servers do not support HTTP compression and
2174    there is Accept-Encoding header in request, haproxy will compress the
2175    matching response.
2176
2177    The "offload" setting makes haproxy remove the Accept-Encoding header to
2178    prevent backend servers from compressing responses. It is strongly
2179    recommended not to do this because this means that all the compression work
2180    will be done on the single point where haproxy is located. However in some
2181    deployment scenarios, haproxy may be installed in front of a buggy gateway
2182    with broken HTTP compression implementation which can't be turned off.
2183    In that case haproxy can be used to prevent that gateway from emitting
2184    invalid payloads. In this case, simply removing the header in the
2185    configuration does not work because it applies before the header is parsed,
2186    so that prevents haproxy from compressing. The "offload" setting should
2187    then be used for such scenarios. Note: for now, the "offload" setting is
2188    ignored when set in a defaults section.
2189
2190    Compression is disabled when:
2191      * the request does not advertise a supported compression algorithm in the
2192        "Accept-Encoding" header
2193      * the response message is not HTTP/1.1
2194      * HTTP status code is not 200
2195      * response header "Transfer-Encoding" contains "chunked" (Temporary
2196        Workaround)
2197      * response contain neither a "Content-Length" header nor a
2198        "Transfer-Encoding" whose last value is "chunked"
2199      * response contains a "Content-Type" header whose first value starts with
2200        "multipart"
2201      * the response contains the "no-transform" value in the "Cache-control"
2202        header
2203      * User-Agent matches "Mozilla/4" unless it is MSIE 6 with XP SP2, or MSIE 7
2204        and later
2205      * The response contains a "Content-Encoding" header, indicating that the
2206        response is already compressed (see compression offload)
2207
2208    Note: The compression does not rewrite Etag headers, and does not emit the
2209          Warning header.
2210
```

```
   Examples :
         compression algo gzip
         compression type text/html text/plain

contimeout <timeout> (deprecated)
  Set the maximum time to wait for a connection attempt to a server to succeed.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <timeout> is the timeout value is specified in milliseconds by default, but
              can be in any other unit if the number is suffixed by the unit,
              as explained at the top of this document.

  If the server is located on the same LAN as haproxy, the connection should be
  immediate (less than a few milliseconds). Anyway, it is a good practice to
  cover one or several TCP packet losses by specifying timeouts that are
  slightly above multiples of 3 seconds (eg: 4 or 5 seconds). By default, the
  connect timeout also presets the queue timeout to the same value if this one
  has not been specified. Historically, the contimeout was also used to set the
  tarpit timeout in a listen section, which is not possible in a pure frontend.

  This parameter is specific to backends, but can be specified once for all in
  "defaults" sections. This is in fact one of the easiest solutions not to
  forget about it. An unspecified timeout results in an infinite timeout, which
  is not recommended. Such a usage is accepted and works but reports a warning
  during startup because it may results in accumulation of failed sessions in
  the system if the system's timeouts are not configured either.

  This parameter is provided for backwards compatibility but is currently
  deprecated. Please use "timeout connect", "timeout queue" or "timeout tarpit"
  instead.

  See also : "timeout connect", "timeout queue", "timeout tarpit",
             "timeout server", "contimeout".


cookie <name> [ rewrite | insert | prefix ] [ indirect ] [ nocache ]
              [ postonly ] [ preserve ] [ httponly ] [ secure ]
              [ domain <domain> ]* [ maxidle <idle> ] [ maxlife <life> ]
  Enable cookie-based persistence in a backend.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <name>    is the name of the cookie which will be monitored, modified or
              inserted in order to bring persistence. This cookie is sent to
              the client via a "Set-Cookie" header in the response, and is
              brought back by the client in a "Cookie" header in all requests.
              Special care should be taken to choose a name which does not
              conflict with any likely application cookie. Also, if the same
              backends are subject to be used by the same clients (eg:
              HTTP/HTTPS), care should be taken to use different cookie names
              between all backends if persistence between them is not desired.

    rewrite   This keyword indicates that the cookie will be provided by the
              server and that haproxy will have to modify its value to set the
              server's identifier in it. This mode is handy when the management
              of complex combinations of "Set-cookie" and "Cache-control"
              headers is left to the application. The application can then
              decide whether or not it is appropriate to emit a persistence
              cookie. Since all responses should be monitored, this mode only
              works in HTTP close mode. Unless the application behaviour is
              very complex and/or broken, it is advised not to start with this
              mode for new deployments. This keyword is incompatible with
              "insert" and "prefix".
```



```
    insert    This keyword indicates that the persistence cookie will have to
              be inserted by haproxy in server responses if the client did not

              already have a cookie that would have permitted it to access this
              server. When used without the "preserve" option, if the server
              emits a cookie with the same name, it will be remove before
              processing.  For this reason, this mode can be used to upgrade
              existing configurations running in the "rewrite" mode. The cookie
              will only be a session cookie and will not be stored on the
              client's disk. By default, unless the "indirect" option is added,
              the server will see the cookies emitted by the client. Due to
              caching effects, it is generally wise to add the "nocache" or
              "postonly" keywords (see below). The "insert" keyword is not
              compatible with "rewrite" and "prefix".

    prefix    This keyword indicates that instead of relying on a dedicated
              cookie for the persistence, an existing one will be completed.
              This may be needed in some specific environments where the client
              does not support more than one single cookie and the application
              already needs it. In this case, whenever the server sets a cookie
              named <name>, it will be prefixed with the server's identifier
              and a delimiter. The prefix will be removed from all client
              requests so that the server still finds the cookie it emitted.
              Since all requests and responses are subject to being modified,
              this mode requires the HTTP close mode. The "prefix" keyword is
              not compatible with "rewrite" and "insert". Note: it is highly
              recommended not to use "indirect" with "prefix", otherwise server
              cookie updates would not be sent to clients.

    indirect  When this option is specified, no cookie will be emitted to a
              client which already has a valid one for the server which has
              processed the request. If the server sets such a cookie itself,
              it will be removed, unless the "preserve" option is also set. In
              "insert" mode, this will additionally remove cookies from the
              requests transmitted to the server, making the persistence
              mechanism totally transparent from an application point of view.
              Note: it is highly recommended not to use "indirect" with
              "prefix", otherwise server cookie updates would not be sent to
              clients.

    nocache   This option is recommended in conjunction with the insert mode
              when there is a cache between the client and HAProxy, as it
              ensures that a cacheable response will be tagged non-cacheable if
              a cookie needs to be inserted. This is important because if all
              persistence cookies are added on a cacheable home page for
              instance, then all customers will then fetch the page from an
              outer cache and will all share the same persistence cookie,
              leading to one server receiving much more traffic than others.
              See also the "insert" and "postonly" options.

    postonly  This option ensures that cookie insertion will only be performed
              on responses to POST requests. It is an alternative to the
              "nocache" option, because POST responses are not cacheable, so
              this ensures that the persistence cookie will never get cached.
              Since most sites do not need any sort of persistence before the
              first POST which generally is a login request, this is a very
              efficient method to optimize caching without risking to find a
              persistence cookie in the cache.
              See also the "insert" and "nocache" options.

    preserve  This option may only be used with "insert" and/or "indirect". It
              allows the server to emit the persistence cookie itself. In this
              case, if a cookie is found in the response, haproxy will leave it
              untouched. This is useful in order to end persistence after a
              logout request for instance. For this, the server just has to
```

```
2341                emit a cookie with an invalid value (eg: empty) or with a date in
2342                the past. By combining this mechanism with the "disable-on-404"
2343                check option, it is possible to perform a completely graceful
2344                shutdown because users will definitely leave the server after
2345                they logout.
2346
2347      httponly  This option tells haproxy to add an "HttpOnly" cookie attribute
2348                when a cookie is inserted. This attribute is used so that a
2349                user agent doesn't share the cookie with non-HTTP components.
2350                Please check RFC6265 for more information on this attribute.
2351
2352      secure    This option tells haproxy to add a "Secure" cookie attribute when
2353                a cookie is inserted. This attribute is used so that a user agent
2354                never emits this cookie over non-secure channels, which means
2355                that a cookie learned with this flag will be presented only over
2356                SSL/TLS connections. Please check RFC6265 for more information on
2357                this attribute.
2358
2359      domain    This option allows to specify the domain at which a cookie is
2360                inserted. It requires exactly one parameter: a valid domain
2361                name. If the domain begins with a dot, the browser is allowed to
2362                use it for any host ending with that name. It is also possible to
2363                specify several domain names by invoking this option multiple
2364                times. Some browsers might have small limits on the number of
2365                domains, so be careful when doing that. For the record, sending
2366                10 domains to MSIE 6 or Firefox 2 works as expected.
2367
2368      maxidle   This option allows inserted cookies to be ignored after some idle
2369                time. It only works with insert-mode cookies. When a cookie is
2370                sent to the client, the date this cookie was emitted is sent too.
2371                Upon further presentations of this cookie, if the date is older
2372                than the delay indicated by the parameter (in seconds), it will
2373                be ignored. Otherwise, it will be refreshed if needed when the
2374                response is sent to the client. This is particularly useful to
2375                prevent users who never close their browsers from remaining for
2376                too long on the same server (eg: after a farm size change). When
2377                this option is set and a cookie has no date, it is always
2378                accepted, but gets refreshed in the response. This maintains the
2379                ability for admins to access their sites. Cookies that have a
2380                date in the future further than 24 hours are ignored. Doing so
2381                lets admins fix timezone issues without risking kicking users off
2382                the site.
2383
2384      maxlife   This option allows inserted cookies to be ignored after some life
2385                time, whether they're in use or not. It only works with insert
2386                mode cookies. When a cookie is first sent to the client, the date
2387                this cookie was emitted is sent too. Upon further presentations
2388                of this cookie, if the date is older than the delay indicated by
2389                the parameter (in seconds), it will be ignored. If the cookie in
2390                the request has no date, it is accepted and a date will be set.
2391                Cookies that have a date in the future further than 24 hours are
2392                ignored. Doing so lets admins fix timezone issues without risking
2393                kicking users off the site. Contrary to maxidle, this value is
2394                not refreshed, only the first visit date counts. Both maxidle and
2395                maxlife may be used at the time. This is particularly useful to
2396                prevent users who never close their browsers from remaining for
2397                too long on the same server (eg: after a farm size change). This
2398                is stronger than the maxidle method in that it forces a
2399                redispatch after some absolute delay.
2400
2401    There can be only one persistence cookie per HTTP backend, and it can be
2402    declared in a defaults section. The value of the cookie will be the value
2403    indicated after the "cookie" keyword in a "server" statement. If no cookie
2404    is declared for a given server, the cookie is not set.
2405
```



```
2406    Examples :
2407          cookie JSESSIONID prefix
2408          cookie SRV insert indirect nocache
2409          cookie SRV insert postonly indirect
2410          cookie SRV insert indirect nocache maxidle 30m maxlife 8h
2411
2412    See also : "appsession", "balance source", "capture cookie", "server"
2413               and "ignore-persist".
2414
2415
2416  default-server [param*]
2417    Change default options for a server in a backend
2418    May be used in sections :   defaults | frontend | listen | backend
2419                                   yes   |    no    |   yes  |   yes
2420    Arguments:
2421      <param*>  is a list of parameters for this server. The "default-server"
2422                keyword accepts an important number of options and has a complete
2423                section dedicated to it. Please refer to section 5 for more
2424                details.
2425
2426    Example :
2427          default-server inter 1000 weight 13
2428
2429    See also: "server" and section 5 about server options
2430
2431
2432  default_backend <backend>
2433    Specify the backend to use when no "use_backend" rule has been matched.
2434    May be used in sections :   defaults | frontend | listen | backend
2435                                   yes   |    yes   |   yes  |   no
2436    Arguments :
2437      <backend> is the name of the backend to use.
2438
2439    When doing content-switching between frontend and backends using the
2440    "use_backend" keyword, it is often useful to indicate which backend will be
2441    used when no rule has matched. It generally is the dynamic backend which
2442    will catch all undetermined requests.
2443
2444    Example :
2445
2446          use_backend     dynamic  if  url_dyn
2447          use_backend     static   if  url_css url_img extension_img
2448          default_backend dynamic
2449
2450    See also : "use_backend", "reqsetbe", "reqisetbe"
2451
2452
2453  description <string>
2454    Describe a listen, frontend or backend.
2455    May be used in sections :   defaults | frontend | listen | backend
2456                                   no    |    yes   |   yes  |   yes
2457    Arguments : string
2458
2459    Allows to add a sentence to describe the related object in the HAProxy HTML
2460    stats page. The description will be printed on the right of the object name
2461    it describes.
2462    No need to backslash spaces in the <string> arguments.
2463
2464
2465  disabled
2466    Disable a proxy, frontend or backend.
2467    May be used in sections :   defaults | frontend | listen | backend
2468                                   yes   |    yes   |   yes  |   yes
2469    Arguments : none
2470
```

```
   The "disabled" keyword is used to disable an instance, mainly in order to
   liberate a listening port or to temporarily disable a service. The instance
   will still be created and its configuration will be checked, but it will be
   created in the "stopped" state and will appear as such in the statistics. It
   will not receive any traffic nor will it send any health-checks or logs. It
   is possible to disable many instances at once by adding the "disabled"
   keyword in a "defaults" section.

   See also : "enabled"


dispatch <address>:<port>
   Set a default server address
   May be used in sections :   defaults | frontend | listen | backend
                                  no    |    no    |   yes  |   yes
   Arguments :

     <address> is the IPv4 address of the default server. Alternatively, a
               resolvable hostname is supported, but this name will be resolved
               during start-up.

     <ports>   is a mandatory port specification. All connections will be sent
               to this port, and it is not permitted to use port offsets as is
               possible with normal servers.

   The "dispatch" keyword designates a default server for use when no other
   server can take the connection. In the past it was used to forward non
   persistent connections to an auxiliary load balancer. Due to its simple
   syntax, it has also been used for simple TCP relays. It is recommended not to
   use it for more clarity, and to use the "server" directive instead.

   See also : "server"


enabled
   Enable a proxy, frontend or backend.
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
   Arguments : none

   The "enabled" keyword is used to explicitly enable an instance, when the
   defaults has been set to "disabled". This is very rarely used.

   See also : "disabled"


errorfile <code> <file>
   Return a file contents instead of errors generated by HAProxy
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
   Arguments :
     <code>    is the HTTP status code. Currently, HAProxy is capable of
               generating codes 200, 400, 403, 408, 500, 502, 503, and 504.

     <file>    designates a file containing the full HTTP response. It is
               recommended to follow the common practice of appending ".http" to
               the filename so that people do not confuse the response with HTML
               error pages, and to use absolute paths, since files are read
               before any chroot is performed.

   It is important to understand that this keyword is not meant to rewrite
   errors returned by the server, but errors detected and returned by HAProxy.
   This is why the list of supported errors is limited to a small set.

   Code 200 is emitted in response to requests matching a "monitor-uri" rule.
```



```

   The files are returned verbatim on the TCP socket. This allows any trick such
   as redirections to another URL or site, as well as tricks to clean cookies,
   force enable or disable caching, etc... The package provides default error
   files returning the same contents as default errors.

   The files should not exceed the configured buffer size (BUFSIZE), which
   generally is 8 or 16 kB, otherwise they will be truncated. It is also wise
   not to put any reference to local contents (eg: images) in order to avoid
   loops between the client and HAProxy when all servers are down, causing an
   error to be returned instead of an image. For better HTTP compliance, it is
   recommended that all header lines end with CR-LF and not LF alone.

   The files are read at the same time as the configuration and kept in memory.
   For this reason, the errors continue to be returned even when the process is
   chrooted, and no file change is considered while the process is running. A
   simple method for developing those files consists in associating them to the
   403 status code and interrogating a blocked URL.

   See also : "errorloc", "errorloc302", "errorloc303"

   Example :
         errorfile 400 /etc/haproxy/errorfiles/400badreq.http
         errorfile 408 /dev/null  # workaround Chrome pre-connect bug
         errorfile 403 /etc/haproxy/errorfiles/403forbid.http
         errorfile 503 /etc/haproxy/errorfiles/503sorry.http


errorloc <code> <url>
errorloc302 <code> <url>
   Return an HTTP redirection to a URL instead of errors generated by HAProxy
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
   Arguments :
     <code>    is the HTTP status code. Currently, HAProxy is capable of
               generating codes 200, 400, 403, 408, 500, 502, 503, and 504.

     <url>     it is the exact contents of the "Location" header. It may contain
               either a relative URI to an error page hosted on the same site,
               or an absolute URI designating an error page on another site.
               Special care should be given to relative URIs to avoid redirect
               loops if the URI itself may generate the same error (eg: 500).

   It is important to understand that this keyword is not meant to rewrite
   errors returned by the server, but errors detected and returned by HAProxy.
   This is why the list of supported errors is limited to a small set.

   Code 200 is emitted in response to requests matching a "monitor-uri" rule.

   Note that both keyword return the HTTP 302 status code, which tells the
   client to fetch the designated URL using the same HTTP method. This can be
   quite problematic in case of non-GET methods such as POST, because the URL
   sent to the client might not be allowed for something other than GET. To
   workaround this problem, please use "errorloc303" which send the HTTP 303
   status code, indicating to the client that the URL must be fetched with a GET
   request.

   See also : "errorfile", "errorloc303"


errorloc303 <code> <url>
   Return an HTTP redirection to a URL instead of errors generated by HAProxy
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
   Arguments :
```

```
      <code>    is the HTTP status code. Currently, HAProxy is capable of
                generating codes 400, 403, 408, 500, 502, 503, and 504.

      <url>     it is the exact contents of the "Location" header. It may contain
                either a relative URI to an error page hosted on the same site,
                or an absolute URI designating an error page on another site.
                Special care should be given to relative URIs to avoid redirect
                loops if the URI itself may generate the same error (eg: 500).

   It is important to understand that this keyword is not meant to rewrite
   errors returned by the server, but errors detected and returned by HAProxy.
   This is why the list of supported errors is limited to a small set.

   Code 200 is emitted in response to requests matching a "monitor-uri" rule.

   Note that both keyword return the HTTP 303 status code, which tells the
   client to fetch the designated URL using the same HTTP GET method. This
   solves the usual problems associated with "errorloc" and the 302 code. It is
   possible that some very old browsers designed before HTTP/1.1 do not support
   it, but no such problem has been reported till now.

   See also : "errorfile", "errorloc", "errorloc302"


force-persist { if | unless } <condition>
  Declare a condition to force persistence on down servers
  May be used in sections:    defaults | frontend | listen | backend
                                  no   |    yes   |   yes  |   yes

  By default, requests are not dispatched to down servers. It is possible to
  force this using "option persist", but it is unconditional and redispatches
  to a valid server if "option redispatch" is set. That leaves with very little
  possibilities to force some requests to reach a server which is artificially
  marked down for maintenance operations.

  The "force-persist" statement allows one to declare various ACL-based
  conditions which, when met, will cause a request to ignore the down status of
  a server and still try to connect to it. That makes it possible to start a
  server, still replying an error to the health checks, and run a specially
  configured browser to test the service. Among the handy methods, one could
  use a specific source IP address, or a specific cookie. The cookie also has
  the advantage that it can easily be added/removed on the browser from a test
  page. Once the service is validated, it is then possible to open the service
  to the world by returning a valid response to health checks.

  The forced persistence is enabled when an "if" condition is met, or unless an
  "unless" condition is met. The final redispatch is always disabled when this
  is used.

  See also : "option redispatch", "ignore-persist", "persist",
             and section 7 about ACL usage.


fullconn <conns>
  Specify at what backend load the servers will reach their maxconn
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <conns>   is the number of connections on the backend which will make the
              servers use the maximal number of connections.

  When a server has a "maxconn" parameter specified, it means that its number
  of concurrent connections will never go higher. Additionally, if it has a
  "minconn" parameter, it indicates a dynamic limit following the backend's
  load. The server will then always accept at least <minconn> connections,
```



```
  never more than <maxconn>, and the limit will be on the ramp between both
  values when the backend has less than <conns> concurrent connections. This
  makes it possible to limit the load on the servers during normal loads, but
  push it further for important loads without overloading the servers during
  exceptional loads.

  Since it's hard to get this value right, haproxy automatically sets it to
  10% of the sum of the maxconns of all frontends that may branch to this
  backend (based on "use_backend" and "default_backend" rules). That way it's
  safe to leave it unset. However, "use_backend" involving dynamic names are
  not counted since there is no way to know if they could match or not.

  Example :
     # The servers will accept between 100 and 1000 concurrent connections each
     # and the maximum of 1000 will be reached when the backend reaches 10000
     # connections.
     backend dynamic
        fullconn   10000
        server     srv1   dyn1:80 minconn 100 maxconn 1000
        server     srv2   dyn2:80 minconn 100 maxconn 1000

  See also : "maxconn", "server"


grace <time>
  Maintain a proxy operational for some time after a soft stop
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments :
    <time>    is the time (by default in milliseconds) for which the instance
              will remain operational with the frontend sockets still listening
              when a soft-stop is received via the SIGUSR1 signal.

  This may be used to ensure that the services disappear in a certain order.
  This was designed so that frontends which are dedicated to monitoring by an
  external equipment fail immediately while other ones remain up for the time
  needed by the equipment to detect the failure.

  Note that currently, there is very little benefit in using this parameter,
  and it may in fact complicate the soft-reconfiguration process more than
  simplify it.


hash-type <method> <function> <modifier>
  Specify a method to use for mapping hashes to servers
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <method> is the method used to select a server from the hash computed by
             the <function> :

      map-based   the hash table is a static array containing all alive servers.
                  The hashes will be very smooth, will consider weights, but
                  will be static in that weight changes while a server is up
                  will be ignored. This means that there will be no slow start.
                  Also, since a server is selected by its position in the array,
                  most mappings are changed when the server count changes. This
                  means that when a server goes up or down, or when a server is
                  added to a farm, most connections will be redistributed to
                  different servers. This can be inconvenient with caches for
                  instance.

      consistent  the hash table is a tree filled with many occurrences of each
                  server. The hash key is looked up in the tree and the closest
                  server is chosen. This hash is dynamic, it supports changing
```

```
                  weights while the servers are up, so it is compatible with the
                  slow start feature. It has the advantage that when a server
                  goes up or down, only its associations are moved. When a
                  server is added to the farm, only a few part of the mappings
                  are redistributed, making it an ideal method for caches.
                  However, due to its principle, the distribution will never be
                  very smooth and it may sometimes be necessary to adjust a
                  server's weight or its ID to get a more balanced distribution.
                  In order to get the same distribution on multiple load
                  balancers, it is important that all servers have the exact
                  same IDs. Note: consistent hash uses sdbm and avalanche if no
                  hash function is specified.

    <function> is the hash function to be used :

       sdbm   this function was created initially for sdbm (a public-domain
              reimplementation of ndbm) database library. It was found to do
              well in scrambling bits, causing better distribution of the keys
              and fewer splits. It also happens to be a good general hashing
              function with good distribution, unless the total server weight
              is a multiple of 64, in which case applying the avalanche
              modifier may help.

       djb2   this function was first proposed by Dan Bernstein many years ago
              on comp.lang.c. Studies have shown that for certain workload this
              function provides a better distribution than sdbm. It generally
              works well with text-based inputs though it can perform extremely
              poorly with numeric-only input or when the total server weight is
              a multiple of 33, unless the avalanche modifier is also used.

       wt6    this function was designed for haproxy while testing other
              functions in the past. It is not as smooth as the other ones, but
              is much less sensible to the input data set or to the number of
              servers. It can make sense as an alternative to sdbm+avalanche or
              djb2+avalanche for consistent hashing or when hashing on numeric
              data such as a source IP address or a visitor identifier in a URL
              parameter.

    <modifier> indicates an optional method applied after hashing the key :

       avalanche   This directive indicates that the result from the hash
                   function above should not be used in its raw form but that
                   a 4-byte full avalanche hash must be applied first. The
                   purpose of this step is to mix the resulting bits from the
                   previous hash in order to avoid any undesired effect when
                   the input contains some limited values or when the number of
                   servers is a multiple of one of the hash's components (64
                   for SDBM, 33 for DJB2). Enabling avalanche tends to make the
                   result less predictable, but it's also not as smooth as when
                   using the original function. Some testing might be needed
                   with some workloads. This hash is one of the many proposed
                   by Bob Jenkins.

  The default hash type is "map-based" and is recommended for most usages. The
  default function is "sdbm", the selection of a function should be based on
  the range of the values being hashed.

  See also : "balance", "server"


http-check disable-on-404
  Enable a maintenance mode upon HTTP/404 response to health-checks
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments : none
```



```

  When this option is set, a server which returns an HTTP code 404 will be
  excluded from further load-balancing, but will still receive persistent
  connections. This provides a very convenient method for Web administrators
  to perform a graceful shutdown of their servers. It is also important to note
  that a server which is detected as failed while it was in this mode will not
  generate an alert, just a notice. If the server responds 2xx or 3xx again, it
  will immediately be reinserted into the farm. The status on the stats page
  reports "NOLB" for a server in this mode. It is important to note that this
  option only works in conjunction with the "httpchk" option. If this option
  is used with "http-check expect", then it has precedence over it so that 404
  responses will still be considered as soft-stop.

  See also : "option httpchk", "http-check expect"


http-check expect [!] <match> <pattern>
  Make HTTP health checks consider response contents or specific status codes
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <match>   is a keyword indicating how to look for a specific pattern in the
              response. The keyword may be one of "status", "rstatus",
              "string", or "rstring". The keyword may be preceded by an
              exclamation mark ("!") to negate the match. Spaces are allowed
              between the exclamation mark and the keyword. See below for more
              details on the supported keywords.

    <pattern> is the pattern to look for. It may be a string or a regular
              expression. If the pattern contains spaces, they must be escaped
              with the usual backslash ('\').

  By default, "option httpchk" considers that response statuses 2xx and 3xx
  are valid, and that others are invalid. When "http-check expect" is used,
  it defines what is considered valid or invalid. Only one "http-check"
  statement is supported in a backend. If a server fails to respond or times
  out, the check obviously fails. The available matches are :

    status <string> : test the exact string match for the HTTP status code.
                      A health check response will be considered valid if the
                      response's status code is exactly this string. If the
                      "status" keyword is prefixed with "!", then the response
                      will be considered invalid if the status code matches.

    rstatus <regex> : test a regular expression for the HTTP status code.
                      A health check response will be considered valid if the
                      response's status code matches the expression. If the
                      "rstatus" keyword is prefixed with "!", then the response
                      will be considered invalid if the status code matches.
                      This is mostly used to check for multiple codes.

    string <string> : test the exact string match in the HTTP response body.
                      A health check response will be considered valid if the
                      response's body contains this exact string. If the
                      "string" keyword is prefixed with "!", then the response
                      will be considered invalid if the body contains this
                      string. This can be used to look for a mandatory word at
                      the end of a dynamic page, or to detect a failure when a
                      specific error appears on the check page (eg: a stack
                      trace).

    rstring <regex> : test a regular expression on the HTTP response body.
                      A health check response will be considered valid if the
                      response's body matches this expression. If the "rstring"
                      keyword is prefixed with "!", then the response will be
```

```
                      considered invalid if the body matches the expression.
                      This can be used to look for a mandatory word at the end
                      of a dynamic page, or to detect a failure when a specific
                      error appears on the check page (eg: a stack trace).

   It is important to note that the responses will be limited to a certain size
   defined by the global "tune.chksize" option, which defaults to 16384 bytes.
   Thus, too large responses may not contain the mandatory pattern when using
   "string" or "rstring". If a large response is absolutely required, it is
   possible to change the default max size by setting the global variable.
   However, it is worth keeping in mind that parsing very large responses can
   waste some CPU cycles, especially when regular expressions are used, and that
   it is always better to focus the checks on smaller resources.

   Also "http-check expect" doesn't support HTTP keep-alive. Keep in mind that it
   will automatically append a "Connection: close" header, meaning that this
   header should not be present in the request provided by "option httpchk".

   Last, if "http-check expect" is combined with "http-check disable-on-404",
   then this last one has precedence when the server responds with 404.

   Examples :
         # only accept status 200 as valid
         http-check expect status 200

         # consider SQL errors as errors
         http-check expect ! string SQL\ Error

         # consider status 5xx only as errors
         http-check expect ! rstatus ^5

         # check that we have a correct hexadecimal tag before /html
         http-check expect rstring <!--tag:[0-9a-f]*</html>

   See also : "option httpchk", "http-check disable-on-404"


http-check send-state
  Enable emission of a state header with HTTP health checks
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments : none

  When this option is set, haproxy will systematically send a special header
  "X-Haproxy-Server-State" with a list of parameters indicating to each server
  how they are seen by haproxy. This can be used for instance when a server is
  manipulated without access to haproxy and the operator needs to know whether
  haproxy still sees it up or not, or if the server is the last one in a farm.

  The header is composed of fields delimited by semi-colons, the first of which
  is a word ("UP", "DOWN", "NOLB"), possibly followed by a number of valid
  checks on the total number before transition, just as appears in the stats
  interface. Next headers are in the form "<variable>=<value>", indicating in
  no specific order some values available in the stats interface :
    - a variable "name", containing the name of the backend followed by a slash
      ("/") then the name of the server. This can be used when a server is
      checked in multiple backends.

    - a variable "node" containing the name of the haproxy node, as set in the
      global "node" variable, otherwise the system's hostname if unspecified.

    - a variable "weight" indicating the weight of the server, a slash ("/")
      and the total weight of the farm (just counting usable servers). This
      helps to know if other servers are available to handle the load when this
      one fails.
```



```

    - a variable "scur" indicating the current number of concurrent connections
      on the server, followed by a slash ("/") then the total number of
      connections on all servers of the same backend.

    - a variable "qcur" indicating the current number of requests in the
      server's queue.

  Example of a header received by the application server :
    >>>  X-Haproxy-Server-State: UP 2/3; name=bck/srv2; node=lb1; weight=1/2; \
           scur=13/22; qcur=0

  See also : "option httpchk", "http-check disable-on-404"

http-request { allow | deny | tarpit | auth [realm <realm>] | redirect <rule> |
               add-header <name> <fmt> | set-header <name> <fmt> |
               del-header <name> | set-nice <nice> | set-log-level <level> |
               replace-header <name> <match-regex> <replace-fmt> |
               replace-value <name> <match-regex> <replace-fmt> |
               set-tos <tos> | set-mark <mark> |
               add-acl(<file name>) <key fmt> |
               del-acl(<file name>) <key fmt> |
               del-map(<file name>) <key fmt> |
               set-map(<file name>) <key fmt> <value fmt>
              }
              [ { if | unless } <condition> ]
  Access control for Layer 7 requests

  May be used in sections:   defaults | frontend | listen | backend
                                no    |    yes   |   yes  |   yes

  The http-request statement defines a set of rules which apply to layer 7
  processing. The rules are evaluated in their declaration order when they are
  met in a frontend, listen or backend section. Any rule may optionally be
  followed by an ACL-based condition, in which case it will only be evaluated
  if the condition is true.

  The first keyword is the rule's action. Currently supported actions include :
    - "allow" : this stops the evaluation of the rules and lets the request
      pass the check. No further "http-request" rules are evaluated.

    - "deny" : this stops the evaluation of the rules and immediately rejects
      the request and emits an HTTP 403 error. No further "http-request" rules
      are evaluated.

    - "tarpit" : this stops the evaluation of the rules and immediately blocks
      the request without responding for a delay specified by "timeout tarpit"
      or "timeout connect" if the former is not set. After that delay, if the
      client is still connected, an HTTP error 500 is returned so that the
      client does not suspect it has been tarpitted. Logs will report the flags
      "PT". The goal of the tarpit rule is to slow down robots during an attack
      when they're limited on the number of concurrent requests. It can be very
      efficient against very dumb robots, and will significantly reduce the
      load on firewalls compared to a "deny" rule. But when facing "correctly"
      developed robots, it can make things worse by forcing haproxy and the
      front firewall to support insane number of concurrent connections.

    - "auth" : this stops the evaluation of the rules and immediately responds
      with an HTTP 401 or 407 error code to invite the user to present a valid
      user name and password. No further "http-request" rules are evaluated. An
      optional "realm" parameter is supported, it sets the authentication realm
      that is returned with the response (typically the application's name).

    - "redirect" : this performs an HTTP redirection based on a redirect rule.
      This is exactly the same as the "redirect" statement except that it
```

inserts a redirect rule which can be processed in the middle of other
"http-request" rules and that these rules use the "log-format" strings.
See the "redirect" keyword for the rule's syntax.

- "add-header" appends an HTTP header field whose name is specified in
<name> and whose value is defined by <fmt> which follows the log-format
rules (see Custom Log Format in section 8.2.4). This is particularly
useful to pass connection-specific information to the server (eg: the
client's SSL certificate), or to combine several headers into one. This
rule is not final, so it is possible to add other similar rules. Note
that header addition is performed immediately, so one rule might reuse
the resulting header from a previous rule.

- "set-header" does the same as "add-header" except that the header name
is first removed if it existed. This is useful when passing security
information to the server, where the header must not be manipulated by
external users. Note that the new value is computed before the removal so
it is possible to concatenate a value to an existing header.

- "del-header" removes all HTTP header fields whose name is specified in
<name>.

- "replace-header" matches the regular expression in all occurrences of
header field <name> according to <match-regex>, and replaces them with
the <replace-fmt> argument. Format characters are allowed in replace-fmt
and work like in <fmt> arguments in "add-header". The match is only
case-sensitive. It is important to understand that this action only
considers whole header lines, regardless of the number of values they
may contain. This usage is suited to headers naturally containing commas
in their value, such as If-Modified-Since and so on.

Example:

```
http-request replace-header Cookie foo=([^;]*);(.*) foo=\1;ip=%bi;\2
```

applied to:

```
Cookie: foo=foobar; expires=Tue, 14-Jun-2016 01:40:45 GMT;
```

outputs:

```
Cookie: foo=foobar;ip=192.168.1.20; expires=Tue, 14-Jun-2016 01:40:45 GMT;
```

assuming the backend IP is 192.168.1.20

- "replace-value" works like "replace-header" except that it matches the
regex against every comma-delimited value of the header field <name>
instead of the entire header. This is suited for all headers which are
allowed to carry more than one value. An example could be the Accept
header.

Example:

```
http-request replace-value X-Forwarded-For ^192\.168\.(.*)$ 172.16.\1
```

applied to:

```
X-Forwarded-For: 192.168.10.1, 192.168.13.24, 10.0.0.37
```

outputs:

```
X-Forwarded-For: 172.16.10.1, 172.16.13.24, 10.0.0.37
```

- "set-nice" sets the "nice" factor of the current request being processed.
It only has effect against the other requests being processed at the same

time. The default value is 0, unless altered by the "nice" setting on the
"bind" line. The accepted range is -1024..1024. The higher the value, the
nicest the request will be. Lower values will make the request more
important than other ones. This can be useful to improve the speed of
some requests, or lower the priority of non-important requests. Using
this setting without prior experimentation can cause some major slowdown.

- "set-log-level" is used to change the log level of the current request
when a certain condition is met. Valid levels are the 8 syslog levels
(see the "log" keyword) plus the special level "silent" which disables
logging for this request. This rule is not final so the last matching
rule wins. This rule can be useful to disable health checks coming from
another equipment.

- "set-tos" is used to set the TOS or DSCP field value of packets sent to
the client to the value passed in <tos> on platforms which support this.
This value represents the whole 8 bits of the IP TOS field, and can be
expressed both in decimal or hexadecimal format (prefixed by "0x"). Note
that only the 6 higher bits are used in DSCP or TOS, and the two lower
bits are always 0. This can be used to adjust some routing behaviour on
border routers based on some information from the request. See RFC 2474,
2597, 3260 and 4594 for more information.

- "set-mark" is used to set the Netfilter MARK on all packets sent to the
client to the value passed in <mark> on platforms which support it. This
value is an unsigned 32 bit value which can be matched by netfilter and
by the routing table. It can be expressed both in decimal or hexadecimal
format (prefixed by "0x"). This can be useful to force certain packets to
take a different route (for example a cheaper network path for bulk
downloads). This works on Linux kernels 2.6.32 and above and requires
admin privileges.

- "add-acl" is used to add a new entry into an ACL. The ACL must be loaded
from a file (even a dummy empty file). The file name of the ACL to be
updated is passed between parentheses. It takes one argument: <key fmt>,
which follows log-format rules, to collect content of the new entry. It
performs a lookup in the ACL before insertion, to avoid duplicated (or
more) values. This lookup is done by a linear search and can be expensive
with large lists! It is the equivalent of the "add acl" command from the
stats socket, but can be triggered by an HTTP request.

- "del-acl" is used to delete an entry from an ACL. The ACL must be loaded
from a file (even a dummy empty file). The file name of the ACL to be
updated is passed between parentheses. It takes one argument: <key fmt>,
which follows log-format rules, to collect content of the entry to delete.
It is the equivalent of the "del acl" command from the stats socket, but
can be triggered by an HTTP request.

- "del-map" is used to delete an entry from a MAP. The MAP must be loaded
from a file (even a dummy empty file). The file name of the MAP to be
updated is passed between parentheses. It takes one argument: <key fmt>,
which follows log-format rules, to collect content of the entry to delete.
It takes one argument: "file name" It is the equivalent of the "del map"
command from the stats socket, but can be triggered by an HTTP request.

- "set-map" is used to add a new entry into a MAP. The MAP must be loaded
from a file (even a dummy empty file). The file name of the MAP to be
updated is passed between parentheses. It takes 2 arguments: <key fmt>,
which follows log-format rules, used to collect MAP key, and <value fmt>,
which follows log-format rules, used to collect content for the new entry.
It performs a lookup in the MAP before insertion, to avoid duplicated (or
more) values. This lookup is done by a linear search and can be expensive
with large lists! It is the equivalent of the "set map" command from the
stats socket, but can be triggered by an HTTP request.

```
   There is no limit to the number of http-request statements per instance.

   It is important to know that http-request rules are processed very early in
   the HTTP processing, just after "block" rules and before "reqdel" or "reqrep"
   rules. That way, headers added by "add-header"/"set-header" are visible by
   almost all further ACL rules.

   Example:
         acl nagios src 192.168.129.3
         acl local_net src 192.168.0.0/16
         acl auth_ok http_auth(L1)

         http-request allow if nagios
         http-request allow if local_net auth_ok
         http-request auth realm Gimme if local_net auth_ok
         http-request deny

   Example:
         acl auth_ok http_auth_group(L1) G1
         http-request auth unless auth_ok

   Example:
         http-request set-header X-Haproxy-Current-Date %T
         http-request set-header X-SSL                  %[ssl_fc]
         http-request set-header X-SSL-Session_ID       %[ssl_fc_session_id,hex]
         http-request set-header X-SSL-Client-Verify    %[ssl_c_verify]
         http-request set-header X-SSL-Client-DN        %{+Q}[ssl_c_s_dn]
         http-request set-header X-SSL-Client-CN        %{+Q}[ssl_c_s_dn(cn)]
         http-request set-header X-SSL-Issuer           %{+Q}[ssl_c_i_dn]
         http-request set-header X-SSL-Client-NotBefore %{+Q}[ssl_c_notbefore]
         http-request set-header X-SSL-Client-NotAfter  %{+Q}[ssl_c_notafter]

   Example:
         acl key req.hdr(X-Add-Acl-Key) -m found
         acl add path /addacl
         acl del path /delacl

         acl myhost hdr(Host) -f myhost.lst

         http-request add-acl(myhost.lst) %[req.hdr(X-Add-Acl-Key)] if key add
         http-request del-acl(myhost.lst) %[req.hdr(X-Add-Acl-Key)] if key del

   Example:
         acl value  req.hdr(X-Value) -m found
         acl setmap path /setmap
         acl delmap path /delmap

         use_backend bk_appli if { hdr(Host),map_str(map.lst) -m found }

         http-request set-map(map.lst) %[src] %[req.hdr(X-Value)] if setmap value
         http-request del-map(map.lst) %[src]                     if delmap

   See also : "stats http-request", section 3.4 about userlists and section 7
              about ACL usage.

http-response { allow | deny | add-header <name> <fmt> | set-nice <nice> |
                set-header <name> <fmt> | del-header <name> |
                replace-header <name> <regex-match> <replace-fmt> |
                replace-value <name> <regex-match> <replace-fmt> |
                set-log-level <level> | set-mark <mark> | set-tos <tos> |
                add-acl(<file name>) <key fmt> |
                del-acl(<file name>) <key fmt> |
                del-map(<file name>) <key fmt> |
                set-map(<file name>) <key fmt> <value fmt>
              }
```



```
              [ { if | unless } <condition> ]
   Access control for Layer 7 responses

   May be used in sections:   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes

   The http-response statement defines a set of rules which apply to layer 7
   processing. The rules are evaluated in their declaration order when they are
   met in a frontend, listen or backend section. Any rule may optionally be
   followed by an ACL-based condition, in which case it will only be evaluated
   if the condition is true. Since these rules apply on responses, the backend
   rules are applied first, followed by the frontend's rules.

   The first keyword is the rule's action. Currently supported actions include :
     - "allow" : this stops the evaluation of the rules and lets the response
       pass the check. No further "http-response" rules are evaluated for the
       current section.

     - "deny" : this stops the evaluation of the rules and immediately rejects
       the response and emits an HTTP 502 error. No further "http-response"
       rules are evaluated.

     - "add-header" appends an HTTP header field whose name is specified in
       <name> and whose value is defined by <fmt> which follows the log-format
       rules (see Custom Log Format in section 8.2.4). This may be used to send
       a cookie to a client for example, or to pass some internal information.
       This rule is not final, so it is possible to add other similar rules.
       Note that header addition is performed immediately, so one rule might
       reuse the resulting header from a previous rule.

     - "set-header" does the same as "add-header" except that the header name
       is first removed if it existed. This is useful when passing security
       information to the server, where the header must not be manipulated by
       external users.

     - "del-header" removes all HTTP header fields whose name is specified in
       <name>.

     - "replace-header" matches the regular expression in all occurrences of
       header field <name> according to <match-regex>, and replaces them with
       the <replace-fmt> argument. Format characters are allowed in replace-fmt
       and work like in <fmt> arguments in "add-header". The match is only
       case-sensitive. It is important to understand that this action only
       considers whole header lines, regardless of the number of values they
       may contain. This usage is suited to headers naturally containing commas
       in their value, such as Set-Cookie, Expires and so on.

       Example:

         http-response replace-header Set-Cookie (C=[^;]*);(.*) \1;ip=%bi;\2

       applied to:

         Set-Cookie: C=1; expires=Tue, 14-Jun-2016 01:40:45 GMT

       outputs:

         Set-Cookie: C=1;ip=192.168.1.20; expires=Tue, 14-Jun-2016 01:40:45 GMT

       assuming the backend IP is 192.168.1.20.

     - "replace-value" works like "replace-header" except that it matches the
       regex against every comma-delimited value of the header field <name>
       instead of the entire header. This is suited for all headers which are
       allowed to carry more than one value. An example could be the Accept
```

```
3251        header.
3252
3253        Example:
3254
3255          http-response replace-value Cache-control ^public$ private
3256
3257        applied to:
3258
3259          Cache-Control: max-age=3600, public
3260
3261        outputs:
3262
3263          Cache-Control: max-age=3600, private
3264
3265      - "set-nice" sets the "nice" factor of the current request being processed.
3266        It only has effect against the other requests being processed at the same
3267        time. The default value is 0, unless altered by the "nice" setting on the
3268        "bind" line. The accepted range is -1024..1024. The higher the value, the
3269        nicest the request will be. Lower values will make the request more
3270        important than other ones. This can be useful to improve the speed of
3271        some requests, or lower the priority of non-important requests. Using
3272        this setting without prior experimentation can cause some major slowdown.
3273
3274      - "set-log-level" is used to change the log level of the current request
3275        when a certain condition is met. Valid levels are the 8 syslog levels
3276        (see the "log" keyword) plus the special level "silent" which disables
3277        logging for this request. This rule is not final so the last matching
3278        rule wins. This rule can be useful to disable health checks coming from
3279        another equipment.
3280
3281      - "set-tos" is used to set the TOS or DSCP field value of packets sent to
3282        the client to the value passed in <tos> on platforms which support this.
3283        This value represents the whole 8 bits of the IP TOS field, and can be
3284        expressed both in decimal or hexadecimal format (prefixed by "0x"). Note
3285        that only the 6 higher bits are used in DSCP or TOS, and the two lower
3286        bits are always 0. This can be used to adjust some routing behaviour on
3287        border routers based on some information from the request. See RFC 2474,
3288        2597, 3260 and 4594 for more information.
3289
3290      - "set-mark" is used to set the Netfilter MARK on all packets sent to the
3291        client to the value passed in <mark> on platforms which support it. This
3292        value is an unsigned 32 bit value which can be matched by netfilter and
3293        by the routing table. It can be expressed both in decimal or hexadecimal
3294        format (prefixed by "0x"). This can be useful to force certain packets to
3295        take a different route (for example a cheaper network path for bulk
3296        downloads). This works on Linux kernels 2.6.32 and above and requires
3297        admin privileges.
3298
3299      - "add-acl" is used to add a new entry into an ACL. The ACL must be loaded
3300        from a file (even a dummy empty file). The file name of the ACL to be
3301        updated is passed between parentheses. It takes one argument: <key fmt>,
3302        which follows log-format rules, to collect content of the new entry. It
3303        performs a lookup in the ACL before insertion, to avoid duplicated (or
3304        more) values. This lookup is done by a linear search and can be expensive
3305        with large lists! It is the equivalent of the "add acl" command from the
3306        stats socket, but can be triggered by an HTTP response.
3307
3308      - "del-acl" is used to delete an entry from an ACL. The ACL must be loaded
3309        from a file (even a dummy empty file). The file name of the ACL to be
3310        updated is passed between parentheses. It takes one argument: <key fmt>,
3311        which follows log-format rules, to collect content of the entry to delete.
3312        It is the equivalent of the "del acl" command from the stats socket, but
3313        can be triggered by an HTTP response.
3314
3315      - "del-map" is used to delete an entry from a MAP. The MAP must be loaded
```



```
3316        from a file (even a dummy empty file). The file name of the MAP to be
3317        updated is passed between parentheses. It takes one argument: <key fmt>,
3318        which follows log-format rules, to collect content of the entry to delete.
3319        It takes one argument: "file name" It is the equivalent of the "del map"
3320        command from the stats socket, but can be triggered by an HTTP response.
3321
3322      - "set-map" is used to add a new entry into a MAP. The MAP must be loaded
3323        from a file (even a dummy empty file). The file name of the MAP to be
3324        updated is passed between parentheses. It takes 2 arguments: <key fmt>,
3325        which follows log-format rules, used to collect MAP key, and <value fmt>,
3326        which follows log-format rules, used to collect content for the new entry.
3327        It performs a lookup in the MAP before insertion, to avoid duplicated (or
3328        more) values. This lookup is done by a linear search and can be expensive
3329        with large lists! It is the equivalent of the "set map" command from the
3330        stats socket, but can be triggered by an HTTP response.
3331
3332    There is no limit to the number of http-response statements per instance.
3333
3334    It is important to know that http-response rules are processed very early in
3335    the HTTP processing, before "reqdel" or "reqrep" rules. That way, headers
3336    added by "add-header"/"set-header" are visible by almost all further ACL
3337    rules.
3338
3339    Example:
3340           acl key_acl res.hdr(X-Acl-Key) -m found
3341
3342           acl myhost hdr(Host) -f myhost.lst
3343
3344           http-response add-acl(myhost.lst) %[res.hdr(X-Acl-Key)] if key_acl
3345           http-response del-acl(myhost.lst) %[res.hdr(X-Acl-Key)] if key_acl
3346
3347    Example:
3348           acl value  res.hdr(X-Value) -m found
3349
3350           use_backend bk_appli if { hdr(Host),map_str(map.lst) -m found }
3351
3352           http-response set-map(map.lst) %[src] %[res.hdr(X-Value)] if value
3353           http-response del-map(map.lst) %[src]                     if ! value
3354
3355    See also : "http-request", section 3.4 about userlists and section 7 about
3356               ACL usage.
3357
3358
3359  http-send-name-header [<header>]
3360    Add the server name to a request. Use the header string given by <header>
3361
3362    May be used in sections:   defaults | frontend | listen | backend
3363                                  yes    |    no    |   yes  |   yes
3364
3365    Arguments :
3366
3367      <header>  The header string to use to send the server name
3368
3369    The "http-send-name-header" statement causes the name of the target
3370    server to be added to the headers of an HTTP request.  The name
3371    is added with the header string proved.
3372
3373    See also : "server"
3374
3375  id <value>
3376    Set a persistent ID to a proxy.
3377    May be used in sections :   defaults | frontend | listen | backend
3378                                   no    |    yes   |   yes  |   yes
3379    Arguments : none
3380
```

```
   Set a persistent ID for the proxy. This ID must be unique and positive.
   An unused ID will automatically be assigned if unset. The first assigned
   value will be 1. This ID is currently only returned in statistics.


ignore-persist { if | unless } <condition>
  Declare a condition to ignore persistence
  May be used in sections:    defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes

  By default, when cookie persistence is enabled, every requests containing
  the cookie are unconditionally persistent (assuming the target server is up
  and running).

  The "ignore-persist" statement allows one to declare various ACL-based
  conditions which, when met, will cause a request to ignore persistence.
  This is sometimes useful to load balance requests for static files, which
  often don't require persistence. This can also be used to fully disable
  persistence for a specific User-Agent (for example, some web crawler bots).

  Combined with "appsession", it can also help reduce HAProxy memory usage, as
  the appsession table won't grow if persistence is ignored.

  The persistence is ignored when an "if" condition is met, or unless an
  "unless" condition is met.

  See also : "force-persist", "cookie", and section 7 about ACL usage.


log global
log <address> [len <length>] <facility> [<level> [<minlevel>]]
no log
  Enable per-instance logging of events and traffic.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes

  Prefix :
    no         should be used when the logger list must be flushed. For example,
               if you don't want to inherit from the default logger list. This
               prefix does not allow arguments.

  Arguments :
    global     should be used when the instance's logging parameters are the
               same as the global ones. This is the most common usage. "global"
               replaces <address>, <facility> and <level> with those of the log
               entries found in the "global" section. Only one "log global"
               statement may be used per instance, and this form takes no other
               parameter.

    <address>  indicates where to send the logs. It takes the same format as
               for the "global" section's logs, and can be one of :

               - An IPv4 address optionally followed by a colon (':') and a UDP
                 port. If no port is specified, 514 is used by default (the
                 standard syslog port).

               - An IPv6 address followed by a colon (':') and optionally a UDP
                 port. If no port is specified, 514 is used by default (the
                 standard syslog port).

               - A filesystem path to a UNIX domain socket, keeping in mind
                 considerations for chroot (be sure the path is accessible
                 inside the chroot) and uid/gid (be sure the path is
                 appropriately writeable).
```



```
               Any part of the address string may reference any number of
               environment variables by preceding their name with a dollar
               sign ('$') and optionally enclosing them with braces ('{}'),
               similarly to what is done in Bourne shell.

    <length>   is an optional maximum line length. Log lines larger than this
               value will be truncated before being sent. The reason is that
               syslog servers act differently on log line length. All servers
               support the default value of 1024, but some servers simply drop
               larger lines while others do log them. If a server supports long
               lines, it may make sense to set this value here in order to avoid
               truncating long lines. Similarly, if a server drops long lines,
               it is preferable to truncate them before sending them. Accepted
               values are 80 to 65535 inclusive. The default value of 1024 is
               generally fine for all standard usages. Some specific cases of
               long captures or JSON-formated logs may require larger values.

    <facility> must be one of the 24 standard syslog facilities :

                 kern   user   mail   daemon auth   syslog lpr    news
                 uucp   cron   auth2  ftp    ntp    audit  alert  cron2
                 local0 local1 local2 local3 local4 local5 local6 local7

    <level>    is optional and can be specified to filter outgoing messages. By
               default, all messages are sent. If a level is specified, only
               messages with a severity at least as important as this level
               will be sent. An optional minimum level can be specified. If it
               is set, logs emitted with a more severe level than this one will
               be capped to this level. This is used to avoid sending "emerg"
               messages on all terminals on some default syslog configurations.
               Eight levels are known :

                 emerg  alert  crit   err    warning notice info  debug

  It is important to keep in mind that it is the frontend which decides what to
  log from a connection, and that in case of content switching, the log entries
  from the backend will be ignored. Connections are logged at level "info".

  However, backend log declaration define how and where servers status changes
  will be logged. Level "notice" will be used to indicate a server going up,
  "warning" will be used for termination signals and definitive service
  termination, and "alert" will be used for when a server goes down.

  Note : According to RFC3164, messages are truncated to 1024 bytes before
         being emitted.

  Example :
    log global
    log 127.0.0.1:514 local0 notice         # only send important events
    log 127.0.0.1:514 local0 notice notice  # same but limit output level
    log ${LOCAL_SYSLOG}:514 local0 notice   # send to local server


log-format <string>
  Specifies the log format string to use for traffic logs
  May be used in sections:    defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |    no

  This directive specifies the log format string that will be used for all logs
  resulting from traffic passing through the frontend using this line. If the
  directive is used in a defaults section, all subsequent frontends will use
  the same log format. Please see section 8.2.4 which covers the log format
  string in depth.
```

```
max-keep-alive-queue <value>
  Set the maximum server queue size for maintaining keep-alive connections
  May be used in sections:    defaults | frontend | listen | backend
                                 yes   |     no   |   yes  |   yes

  HTTP keep-alive tries to reuse the same server connection whenever possible,
  but sometimes it can be counter-productive, for example if a server has a lot
  of connections while other ones are idle. This is especially true for static
  servers.

  The purpose of this setting is to set a threshold on the number of queued
  connections at which haproxy stops trying to reuse the same server and prefers
  to find another one. The default value, -1, means there is no limit. A value
  of zero means that keep-alive requests will never be queued. For very close
  servers which can be reached with a low latency and which are not sensible to
  breaking keep-alive, a low value is recommended (eg: local static server can
  use a value of 10 or less). For remote servers suffering from a high latency,
  higher values might be needed to cover for the latency and/or the cost of
  picking a different server.

  Note that this has no impact on responses which are maintained to the same
  server consecutively to a 401 response. They will still go to the same server
  even if they have to be queued.

  See also : "option http-server-close", "option prefer-last-server", server
             "maxconn" and cookie persistence.


maxconn <conns>
  Fix the maximum number of concurrent connections on a frontend
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments :
    <conns>   is the maximum number of concurrent connections the frontend will
              accept to serve. Excess connections will be queued by the system
              in the socket's listen queue and will be served once a connection
              closes.

  If the system supports it, it can be useful on big sites to raise this limit
  very high so that haproxy manages connection queues, instead of leaving the
  clients with unanswered connection attempts. This value should not exceed the
  global maxconn. Also, keep in mind that a connection contains two buffers
  of 8kB each, as well as some other data resulting in about 17 kB of RAM being
  consumed per established connection. That means that a medium system equipped
  with 1GB of RAM can withstand around 40000-50000 concurrent connections if
  properly tuned.

  Also, when <conns> is set to large values, it is possible that the servers
  are not sized to accept such loads, and for this reason it is generally wise
  to assign them some reasonable connection limits.

  By default, this value is set to 2000.

  See also : "server", global section's "maxconn", "fullconn"


mode { tcp|http|health }
  Set the running mode or protocol of the instance
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments :
    tcp       The instance will work in pure TCP mode. A full-duplex connection
              will be established between clients and servers, and no layer 7
              examination will be performed. This is the default mode. It
              should be used for SSL, SSH, SMTP, ...
```



```

    http      The instance will work in HTTP mode. The client request will be
              analyzed in depth before connecting to any server. Any request
              which is not RFC-compliant will be rejected. Layer 7 filtering,
              processing and switching will be possible. This is the mode which
              brings HAProxy most of its value.

    health    The instance will work in "health" mode. It will just reply "OK"
              to incoming connections and close the connection. Alternatively,
              If the "httpchk" option is set, "HTTP/1.0 200 OK" will be sent
              instead. Nothing will be logged in either case. This mode is used
              to reply to external components health checks. This mode is
              deprecated and should not be used anymore as it is possible to do
              the same and even better by combining TCP or HTTP modes with the
              "monitor" keyword.

  When doing content switching, it is mandatory that the frontend and the
  backend are in the same mode (generally HTTP), otherwise the configuration
  will be refused.

  Example :
     defaults http_instances
         mode http

  See also : "monitor", "monitor-net"


monitor fail { if | unless } <condition>
  Add a condition to report a failure to a monitor HTTP request.
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   no
  Arguments :
    if <cond>     the monitor request will fail if the condition is satisfied,
                  and will succeed otherwise. The condition should describe a
                  combined test which must induce a failure if all conditions
                  are met, for instance a low number of servers both in a
                  backend and its backup.

    unless <cond> the monitor request will succeed only if the condition is
                  satisfied, and will fail otherwise. Such a condition may be
                  based on a test on the presence of a minimum number of active
                  servers in a list of backends.

  This statement adds a condition which can force the response to a monitor
  request to report a failure. By default, when an external component queries
  the URI dedicated to monitoring, a 200 response is returned. When one of the
  conditions above is met, haproxy will return 503 instead of 200. This is
  very useful to report a site failure to an external component which may base
  routing advertisements between multiple sites on the availability reported by
  haproxy. In this case, one would rely on an ACL involving the "nbsrv"
  criterion. Note that "monitor fail" only works in HTTP mode. Both status
  messages may be tweaked using "errorfile" or "errorloc" if needed.

  Example:
     frontend www
        mode http
        acl site_dead nbsrv(dynamic) lt 2
        acl site_dead nbsrv(static)  lt 2
        monitor-uri   /site_alive
        monitor fail  if site_dead

  See also : "monitor-net", "monitor-uri", "errorfile", "errorloc"


monitor-net <source>
```

```
3641    Declare a source network which is limited to monitor requests
3642    May be used in sections :   defaults | frontend | listen | backend
3643                                   yes   |    yes   |   yes  |   no
3644    Arguments :
3645      <source>  is the source IPv4 address or network which will only be able to
3646                get monitor responses to any request. It can be either an IPv4
3647                address, a host name, or an address followed by a slash ('/')
3648                followed by a mask.
3649
3650    In TCP mode, any connection coming from a source matching <source> will cause
3651    the connection to be immediately closed without any log. This allows another
3652    equipment to probe the port and verify that it is still listening, without
3653    forwarding the connection to a remote server.
3654
3655    In HTTP mode, a connection coming from a source matching <source> will be
3656    accepted, the following response will be sent without waiting for a request,
3657    then the connection will be closed : "HTTP/1.0 200 OK". This is normally
3658    enough for any front-end HTTP probe to detect that the service is UP and
3659    running without forwarding the request to a backend server. Note that this
3660    response is sent in raw format, without any transformation. This is important
3661    as it means that it will not be SSL-encrypted on SSL listeners.
3662
3663    Monitor requests are processed very early, just after tcp-request connection
3664    ACLs which are the only ones able to block them. These connections are short
3665    lived and never wait for any data from the client. They cannot be logged, and
3666    it is the intended purpose. They are only used to report HAProxy's health to
3667    an upper component, nothing more. Please note that "monitor fail" rules do
3668    not apply to connections intercepted by "monitor-net".
3669
3670    Last, please note that only one "monitor-net" statement can be specified in
3671    a frontend. If more than one is found, only the last one will be considered.
3672
3673    Example :
3674      # addresses .252 and .253 are just probing us.
3675      frontend www
3676          monitor-net 192.168.0.252/31
3677
3678    See also : "monitor fail", "monitor-uri"
3679
3680
3681  monitor-uri <uri>
3682    Intercept a URI used by external components' monitor requests
3683    May be used in sections :   defaults | frontend | listen | backend
3684                                   yes   |    yes   |   yes  |   no
3685    Arguments :
3686      <uri>     is the exact URI which we want to intercept to return HAProxy's
3687                health status instead of forwarding the request.
3688
3689    When an HTTP request referencing <uri> will be received on a frontend,
3690    HAProxy will not forward it nor log it, but instead will return either
3691    "HTTP/1.0 200 OK" or "HTTP/1.0 503 Service unavailable", depending on failure
3692    conditions defined with "monitor fail". This is normally enough for any
3693    front-end HTTP probe to detect that the service is UP and running without
3694    forwarding the request to a backend server. Note that the HTTP method, the
3695    version and all headers are ignored, but the request must at least be valid
3696    at the HTTP level. This keyword may only be used with an HTTP-mode frontend.
3697
3698    Monitor requests are processed very early. It is not possible to block nor
3699    divert them using ACLs. They cannot be logged either, and it is the intended
3700    purpose. They are only used to report HAProxy's health to an upper component,
3701    nothing more. However, it is possible to add any number of conditions using
3702    "monitor fail" and ACLs so that the result can be adjusted to whatever check
3703    can be imagined (most often the number of available servers in a backend).
3704
3705    Example :
```



```
3706      # Use /haproxy_test to report haproxy's status
3707      frontend www
3708          mode http
3709          monitor-uri /haproxy_test
3710
3711    See also : "monitor fail", "monitor-net"
3712
3713
3714  option abortonclose
3715  no option abortonclose
3716    Enable or disable early dropping of aborted requests pending in queues.
3717    May be used in sections :   defaults | frontend | listen | backend
3718                                   yes   |     no   |   yes  |   yes
3719    Arguments : none
3720
3721    In presence of very high loads, the servers will take some time to respond.
3722    The per-instance connection queue will inflate, and the response time will
3723    increase respective to the size of the queue times the average per-session
3724    response time. When clients will wait for more than a few seconds, they will
3725    often hit the "STOP" button on their browser, leaving a useless request in
3726    the queue, and slowing down other users, and the servers as well, because the
3727    request will eventually be served, then aborted at the first error
3728    encountered while delivering the response.
3729
3730    As there is no way to distinguish between a full STOP and a simple output
3731    close on the client side, HTTP agents should be conservative and consider
3732    that the client might only have closed its output channel while waiting for
3733    the response. However, this introduces risks of congestion when lots of users
3734    do the same, and is completely useless nowadays because probably no client at
3735    all will close the session while waiting for the response. Some HTTP agents
3736    support this behaviour (Squid, Apache, HAProxy), and others do not (TUX, most
3737    hardware-based load balancers). So the probability for a closed input channel
3738    to represent a user hitting the "STOP" button is close to 100%, and the risk
3739    of being the single component to break rare but valid traffic is extremely
3740    low, which adds to the temptation to be able to abort a session early while
3741    still not served and not pollute the servers.
3742
3743    In HAProxy, the user can choose the desired behaviour using the option
3744    "abortonclose". By default (without the option) the behaviour is HTTP
3745    compliant and aborted requests will be served. But when the option is
3746    specified, a session with an incoming channel closed will be aborted while
3747    it is still possible, either pending in the queue for a connection slot, or
3748    during the connection establishment if the server has not yet acknowledged
3749    the connection request. This considerably reduces the queue size and the load
3750    on saturated servers when users are tempted to click on STOP, which in turn
3751    reduces the response time for other users.
3752
3753    If this option has been enabled in a "defaults" section, it can be disabled
3754    in a specific instance by prepending the "no" keyword before it.
3755
3756    See also : "timeout queue" and server's "maxconn" and "maxqueue" parameters
3757
3758
3759  option accept-invalid-http-request
3760  no option accept-invalid-http-request
3761    Enable or disable relaxing of HTTP request parsing
3762    May be used in sections :   defaults | frontend | listen | backend
3763                                   yes   |    yes   |   yes  |   no
3764    Arguments : none
3765
3766    By default, HAProxy complies with RFC7230 in terms of message parsing. This
3767    means that invalid characters in header names are not permitted and cause an
3768    error to be returned to the client. This is the desired behaviour as such
3769    forbidden characters are essentially used to build attacks exploiting server
3770    weaknesses, and bypass security filtering. Sometimes, a buggy browser or
```

server will emit invalid header names for whatever reason (configuration, implementation) and the issue will not be immediately fixed. In such a case, it is possible to relax HAProxy's header name parser to accept any character even if that does not make sense, by specifying this option. Similarly, the list of characters allowed to appear in a URI is well defined by RFC3986, and chars 0-31, 32 (space), 34 ('"'), 60 ('<'), 62 ('>'), 92 ('\'), 94 ('^'), 96 ('`'), 123 ('{'), 124 ('|'), 125 ('}'), 127 (delete) and anything above are not allowed at all. Haproxy always blocks a number of them (0..32, 127). The remaining ones are blocked by default unless this option is enabled. This option also relaxes the test on the HTTP version format, it allows multiple digits for both the major and the minor version.

This option should never be enabled by default as it hides application bugs and open security breaches. It should only be deployed after a problem has been confirmed.

When this option is enabled, erroneous header names will still be accepted in requests, but the complete request will be captured in order to permit later analysis using the "show errors" request on the UNIX stats socket. Similarly, requests containing invalid chars in the URI part will be logged. Doing this also helps confirming that the issue has been solved.

If this option has been enabled in a "defaults" section, it can be disabled in a specific instance by prepending the "no" keyword before it.

See also : "option accept-invalid-http-response" and "show errors" on the stats socket.

option accept-invalid-http-response
no option accept-invalid-http-response
Enable or disable relaxing of HTTP response parsing
May be used in sections : defaults | frontend | listen | backend
yes | no | yes | yes
Arguments : none

By default, HAProxy complies with RFC7230 in terms of message parsing. This means that invalid characters in header names are not permitted and cause an error to be returned to the client. This is the desired behaviour as such forbidden characters are essentially used to build attacks exploiting server weaknesses, and bypass security filtering. Sometimes, a buggy browser or server will emit invalid header names for whatever reason (configuration, implementation) and the issue will not be immediately fixed. In such a case, it is possible to relax HAProxy's header name parser to accept any character even if that does not make sense, by specifying this option. This option also relaxes the test on the HTTP version format, it allows multiple digits for both the major and the minor version.

This option should never be enabled by default as it hides application bugs and open security breaches. It should only be deployed after a problem has been confirmed.

When this option is enabled, erroneous header names will still be accepted in responses, but the complete response will be captured in order to permit later analysis using the "show errors" request on the UNIX stats socket. Doing this also helps confirming that the issue has been solved.

If this option has been enabled in a "defaults" section, it can be disabled in a specific instance by prepending the "no" keyword before it.

See also : "option accept-invalid-http-request" and "show errors" on the stats socket.

option allbackups

no option allbackups
Use either all backup servers at a time or only the first one
May be used in sections : defaults | frontend | listen | backend
yes | no | yes | yes
Arguments : none

By default, the first operational backup server gets all traffic when normal servers are all down. Sometimes, it may be preferred to use multiple backups at once, because one will not be enough. When "option allbackups" is enabled, the load balancing will be performed among all backup servers when all normal ones are unavailable. The same load balancing algorithm will be used and the servers' weights will be respected. Thus, there will not be any priority order between the backup servers anymore.

This option is mostly used with static server farms dedicated to return a "sorry" page when an application is completely offline.

If this option has been enabled in a "defaults" section, it can be disabled in a specific instance by prepending the "no" keyword before it.

option checkcache
no option checkcache
Analyze all server responses and block responses with cacheable cookies
May be used in sections : defaults | frontend | listen | backend
yes | no | yes | yes
Arguments : none

Some high-level frameworks set application cookies everywhere and do not always let enough control to the developer to manage how the responses should be cached. When a session cookie is returned on a cacheable object, there is a high risk of session crossing or stealing between users traversing the same caches. In some situations, it is better to block the response than to let some sensitive session information go in the wild.

The option "checkcache" enables deep inspection of all server responses for strict compliance with HTTP specification in terms of cacheability. It carefully checks "Cache-control", "Pragma" and "Set-cookie" headers in server response to check if there's a risk of caching a cookie on a client-side proxy. When this option is enabled, the only responses which can be delivered to the client are :

- all those without "Set-Cookie" header ;
- all those with a return code other than 200, 203, 206, 300, 301, 410, provided that the server has not set a "Cache-control: public" header ;
- all those that come from a POST request, provided that the server has not set a 'Cache-Control: public' header ;
- those with a 'Pragma: no-cache' header
- those with a 'Cache-control: private' header
- those with a 'Cache-control: no-store' header
- those with a 'Cache-control: max-age=0' header
- those with a 'Cache-control: s-maxage=0' header
- those with a 'Cache-control: no-cache' header
- those with a 'Cache-control: no-cache="set-cookie"' header
- those with a 'Cache-control: no-cache="set-cookie,' header (allowing other fields after set-cookie)

If a response doesn't respect these requirements, then it will be blocked just as if it was from an "rspdeny" filter, with an "HTTP 502 bad gateway". The session state shows "PH--" meaning that the proxy blocked the response during headers processing. Additionally, an alert will be sent in the logs so that admins are informed that there's something to be fixed.

Due to the high impact on the application, the application should be tested in depth with the option enabled before going to production. It is also a good practice to always activate it during tests, even if it is not used in

```
   production, as it will report potentially dangerous application behaviours.

   If this option has been enabled in a "defaults" section, it can be disabled
   in a specific instance by prepending the "no" keyword before it.


option clitcpka
no option clitcpka
  Enable or disable the sending of TCP keepalive packets on the client side
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments : none

  When there is a firewall or any session-aware component between a client and
  a server, and when the protocol involves very long sessions with long idle
  periods (eg: remote desktops), there is a risk that one of the intermediate
  components decides to expire a session which has remained idle for too long.

  Enabling socket-level TCP keep-alives makes the system regularly send packets
  to the other end of the connection, leaving it active. The delay between
  keep-alive probes is controlled by the system only and depends both on the
  operating system and its tuning parameters.

  It is important to understand that keep-alive packets are neither emitted nor
  received at the application level. It is only the network stacks which sees
  them. For this reason, even if one side of the proxy already uses keep-alives
  to maintain its connection alive, those keep-alive packets will not be
  forwarded to the other side of the proxy.

  Please note that this has nothing to do with HTTP keep-alive.

  Using option "clitcpka" enables the emission of TCP keep-alive probes on the
  client side of a connection, which should help when session expirations are
  noticed between HAProxy and a client.

  If this option has been enabled in a "defaults" section, it can be disabled
  in a specific instance by prepending the "no" keyword before it.

  See also : "option srvtcpka", "option tcpka"


option contstats
  Enable continuous traffic statistics updates
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments : none

  By default, counters used for statistics calculation are incremented
  only when a session finishes. It works quite well when serving small
  objects, but with big ones (for example large images or archives) or
  with A/V streaming, a graph generated from haproxy counters looks like
  a hedgehog. With this option enabled counters get incremented continuously,
  during a whole session. Recounting touches a hotpath directly so
  it is not enabled by default, as it has small performance impact (~0.5%).


option dontlog-normal
no option dontlog-normal
  Enable or disable logging of normal, successful connections
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments : none

  There are large sites dealing with several thousand connections per second
  and for which logging is a major pain. Some of them are even forced to turn
```



```
  logs off and cannot debug production issues. Setting this option ensures that
  normal connections, those which experience no error, no timeout, no retry nor
  redispatch, will not be logged. This leaves disk space for anomalies. In HTTP
  mode, the response status code is checked and return codes 5xx will still be
  logged.

  It is strongly discouraged to use this option as most of the time, the key to
  complex issues is in the normal logs which will not be logged here. If you
  need to separate logs, see the "log-separate-errors" option instead.

  See also : "log", "dontlognull", "log-separate-errors" and section 8 about
             logging.


option dontlognull
no option dontlognull
  Enable or disable logging of null connections
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments : none

  In certain environments, there are components which will regularly connect to
  various systems to ensure that they are still alive. It can be the case from
  another load balancer as well as from monitoring systems. By default, even a
  simple port probe or scan will produce a log. If those connections pollute
  the logs too much, it is possible to enable option "dontlognull" to indicate
  that a connection on which no data has been transferred will not be logged,
  which typically corresponds to those probes. Note that errors will still be
  returned to the client and accounted for in the stats. If this is not what is
  desired, option http-ignore-probes can be used instead.

  It is generally recommended not to use this option in uncontrolled
  environments (eg: internet), otherwise scans and other malicious activities
  would not be logged.

  If this option has been enabled in a "defaults" section, it can be disabled
  in a specific instance by prepending the "no" keyword before it.

  See also : "log", "http-ignore-probes", "monitor-net", "monitor-uri", and
             section 8 about logging.


option forceclose
no option forceclose
  Enable or disable active connection closing after response is transferred.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments : none

  Some HTTP servers do not necessarily close the connections when they receive
  the "Connection: close" set by "option httpclose", and if the client does not
  close either, then the connection remains open till the timeout expires. This
  causes high number of simultaneous connections on the servers and shows high
  global session times in the logs.

  When this happens, it is possible to use "option forceclose". It will
  actively close the outgoing server channel as soon as the server has finished
  to respond and release some resources earlier than with "option httpclose".

  This option may also be combined with "option http-pretend-keepalive", which
  will disable sending of the "Connection: close" header, but will still cause
  the connection to be closed once the whole response is received.

  This option disables and replaces any previous "option httpclose", "option
  http-server-close", "option http-keep-alive", or "option http-tunnel".
```

```
4031
4032    If this option has been enabled in a "defaults" section, it can be disabled
4033    in a specific instance by prepending the "no" keyword before it.
4034
4035    See also : "option httpclose" and "option http-pretend-keepalive"
4036
4037
4038  option forwardfor [ except <network> ] [ header <name> ] [ if-none ]
4039    Enable insertion of the X-Forwarded-For header to requests sent to servers
4040    May be used in sections :   defaults | frontend | listen | backend
4041                                   yes   |    yes   |   yes  |   yes
4042    Arguments :
4043      <network> is an optional argument used to disable this option for sources
4044                matching <network>
4045      <name>    an optional argument to specify a different "X-Forwarded-For"
4046                header name.
4047
4048    Since HAProxy works in reverse-proxy mode, the servers see its IP address as
4049    their client address. This is sometimes annoying when the client's IP address
4050    is expected in server logs. To solve this problem, the well-known HTTP header
4051    "X-Forwarded-For" may be added by HAProxy to all requests sent to the server.
4052    This header contains a value representing the client's IP address. Since this
4053    header is always appended at the end of the existing header list, the server
4054    must be configured to always use the last occurrence of this header only. See
4055    the server's manual to find how to enable use of this standard header. Note
4056    that only the last occurrence of the header must be used, since it is really
4057    possible that the client has already brought one.
4058
4059    The keyword "header" may be used to supply a different header name to replace
4060    the default "X-Forwarded-For". This can be useful where you might already
4061    have a "X-Forwarded-For" header from a different application (eg: stunnel),
4062    and you need preserve it. Also if your backend server doesn't use the
4063    "X-Forwarded-For" header and requires different one (eg: Zeus Web Servers
4064    require "X-Cluster-Client-IP").
4065
4066    Sometimes, a same HAProxy instance may be shared between a direct client
4067    access and a reverse-proxy access (for instance when an SSL reverse-proxy is
4068    used to decrypt HTTPS traffic). It is possible to disable the addition of the
4069    header for a known source address or network by adding the "except" keyword
4070    followed by the network address. In this case, any source IP matching the
4071    network will not cause an addition of this header. Most common uses are with
4072    private networks or 127.0.0.1.
4073
4074    Alternatively, the keyword "if-none" states that the header will only be
4075    added if it is not present. This should only be used in perfectly trusted
4076    environment, as this might cause a security issue if headers reaching haproxy
4077    are under the control of the end-user.
4078
4079    This option may be specified either in the frontend or in the backend. If at
4080    least one of them uses it, the header will be added. Note that the backend's
4081    setting of the header subargument takes precedence over the frontend's if
4082    both are defined. In the case of the "if-none" argument, if at least one of
4083    the frontend or the backend does not specify it, it wants the addition to be
4084    mandatory, so it wins.
4085
4086    Examples :
4087      # Public HTTP address also used by stunnel on the same machine
4088      frontend www
4089          mode http
4090          option forwardfor except 127.0.0.1  # stunnel already adds the header
4091
4092      # Those servers want the IP Address in X-Client
4093      backend www
4094          mode http
4095          option forwardfor header X-Client
```



```
4096
4097    See also : "option httpclose", "option http-server-close",
4098               "option forceclose", "option http-keep-alive"
4099
4100
4101  option http-ignore-probes
4102  no option http-ignore-probes
4103    Enable or disable logging of null connections and request timeouts
4104    May be used in sections :   defaults | frontend | listen | backend
4105                                   yes   |    yes   |   yes  |   no
4106    Arguments : none
4107
4108    Recently some browsers started to implement a "pre-connect" feature
4109    consisting in speculatively connecting to some recently visited web sites
4110    just in case the user would like to visit them. This results in many
4111    connections being established to web sites, which end up in 408 Request
4112    Timeout if the timeout strikes first, or 400 Bad Request when the browser
4113    decides to close them first. These ones pollute the log and feed the error
4114    counters. There was already "option dontlognull" but it's insufficient in
4115    this case. Instead, this option does the following things :
4116       - prevent any 400/408 message from being sent to the client if nothing
4117         was received over a connection before it was closed ;
4118       - prevent any log from being emitted in this situation ;
4119       - prevent any error counter from being incremented
4120
4121    That way the empty connection is silently ignored. Note that it is better
4122    not to use this unless it is clear that it is needed, because it will hide
4123    real problems. The most common reason for not receiving a request and seeing
4124    a 408 is due to an MTU inconsistency between the client and an intermediary
4125    element such as a VPN, which blocks too large packets. These issues are
4126    generally seen with POST requests as well as GET with large cookies. The logs
4127    are often the only way to detect them.
4128
4129    If this option has been enabled in a "defaults" section, it can be disabled
4130    in a specific instance by prepending the "no" keyword before it.
4131
4132    See also : "log", "dontlognull", "errorfile", and section 8 about logging.
4133
4134
4135  option http-keep-alive
4136  no option http-keep-alive
4137    Enable or disable HTTP keep-alive from client to server
4138    May be used in sections :   defaults | frontend | listen | backend
4139                                   yes   |    yes   |   yes  |   yes
4140    Arguments : none
4141
4142    By default HAProxy operates in keep-alive mode with regards to persistent
4143    connections: for each connection it processes each request and response, and
4144    leaves the connection idle on both sides between the end of a response and the
4145    start of a new request. This mode may be changed by several options such as
4146    "option http-server-close", "option forceclose", "option httpclose" or
4147    "option http-tunnel". This option allows to set back the keep-alive mode,
4148    which can be useful when another mode was used in a defaults section.
4149
4150    Setting "option http-keep-alive" enables HTTP keep-alive mode on the client-
4151    and server- sides. This provides the lowest latency on the client side (slow
4152    network) and the fastest session reuse on the server side at the expense
4153    of maintaining idle connections to the servers. In general, it is possible
4154    with this option to achieve approximately twice the request rate that the
4155    "http-server-close" option achieves on small objects. There are mainly two
4156    situations where this option may be useful :
4157
4158      - when the server is non-HTTP compliant and authenticates the connection
4159        instead of requests (eg: NTLM authentication)
4160
```

```
      - when the cost of establishing the connection to the server is significant
        compared to the cost of retrieving the associated object from the server.

   This last case can happen when the server is a fast static server of cache.
   In this case, the server will need to be properly tuned to support high enough
   connection counts because connections will last until the client sends another
   request.

   If the client request has to go to another backend or another server due to
   content switching or the load balancing algorithm, the idle connection will
   immediately be closed and a new one re-opened. Option "prefer-last-server" is
   available to try optimize server selection so that if the server currently
   attached to an idle connection is usable, it will be used.

   In general it is preferred to use "option http-server-close" with application
   servers, and some static servers might benefit from "option http-keep-alive".

   At the moment, logs will not indicate whether requests came from the same
   session or not. The accept date reported in the logs corresponds to the end
   of the previous request, and the request time corresponds to the time spent
   waiting for a new request. The keep-alive request time is still bound to the
   timeout defined by "timeout http-keep-alive" or "timeout http-request" if
   not set.

   This option disables and replaces any previous "option httpclose", "option
   http-server-close", "option forceclose" or "option http-tunnel". When backend
   and frontend options differ, all of these 4 options have precedence over
   "option http-keep-alive".

   See also : "option forceclose", "option http-server-close",
              "option prefer-last-server", "option http-pretend-keepalive",
              "option httpclose", and "1.1. The HTTP transaction model".


option http-no-delay
no option http-no-delay
  Instruct the system to favor low interactive delays over performance in HTTP
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments : none

  In HTTP, each payload is unidirectional and has no notion of interactivity.
  Any agent is expected to queue data somewhat for a reasonably low delay.
  There are some very rare server-to-server applications that abuse the HTTP
  protocol and expect the payload phase to be highly interactive, with many
  interleaved data chunks in both directions within a single request. This is
  absolutely not supported by the HTTP specification and will not work across
  most proxies or servers. When such applications attempt to do this through
  haproxy, it works but they will experience high delays due to the network
  optimizations which favor performance by instructing the system to wait for
  enough data to be available in order to only send full packets. Typical
  delays are around 200 ms per round trip. Note that this only happens with
  abnormal uses. Normal uses such as CONNECT requests nor WebSockets are not
  affected.

  When "option http-no-delay" is present in either the frontend or the backend
  used by a connection, all such optimizations will be disabled in order to
  make the exchanges as fast as possible. Of course this offers no guarantee on
  the functionality, as it may break at any other place. But if it works via
  HAProxy, it will work as fast as possible. This option should never be used
  by default, and should never be used at all unless such a buggy application
  is discovered. The impact of using this option is an increase of bandwidth
  usage and CPU usage, which may significantly lower performance in high
  latency environments.

```



```

option http-pretend-keepalive
no option http-pretend-keepalive
  Define whether haproxy will announce keepalive to the server or not
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments : none

  When running with "option http-server-close" or "option forceclose", haproxy
  adds a "Connection: close" header to the request forwarded to the server.
  Unfortunately, when some servers see this header, they automatically refrain
  from using the chunked encoding for responses of unknown length, while this
  is totally unrelated. The immediate effect is that this prevents haproxy from
  maintaining the client connection alive. A second effect is that a client or
  a cache could receive an incomplete response without being aware of it, and
  consider the response complete.

  By setting "option http-pretend-keepalive", haproxy will make the server
  believe it will keep the connection alive. The server will then not fall back
  to the abnormal undesired above. When haproxy gets the whole response, it
  will close the connection with the server just as it would do with the
  "forceclose" option. That way the client gets a normal response and the
  connection is correctly closed on the server side.

  It is recommended not to enable this option by default, because most servers
  will more efficiently close the connection themselves after the last packet,
  and release its buffers slightly earlier. Also, the added packet on the
  network could slightly reduce the overall peak performance. However it is
  worth noting that when this option is enabled, haproxy will have slightly
  less work to do. So if haproxy is the bottleneck on the whole architecture,
  enabling this option might save a few CPU cycles.

  This option may be set both in a frontend and in a backend. It is enabled if
  at least one of the frontend or backend holding a connection has it enabled.
  This option may be combined with "option httpclose", which will cause
  keepalive to be announced to the server and close to be announced to the
  client. This practice is discouraged though.

  If this option has been enabled in a "defaults" section, it can be disabled
  in a specific instance by prepending the "no" keyword before it.

  See also : "option forceclose", "option http-server-close", and
             "option http-keep-alive"


option http-server-close
no option http-server-close
  Enable or disable HTTP connection closing on the server side
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments : none

  By default HAProxy operates in keep-alive mode with regards to persistent
  connections: for each connection it processes each request and response, and
  leaves the connection idle on both sides between the end of a response and
  the start of a new request. This mode may be changed by several options such
  as "option http-server-close", "option forceclose", "option httpclose" or
  "option http-tunnel". Setting "option http-server-close" enables HTTP
  connection-close mode on the server side while keeping the ability to support
  HTTP keep-alive and pipelining on the client side.  This provides the lowest
  latency on the client side (slow network) and the fastest session reuse on
  the server side to save server resources, similarly to "option forceclose".
  It also permits non-keepalive capable servers to be served in keep-alive mode
  to the clients if they conform to the requirements of RFC2616. Please note
  that some servers do not always conform to those requirements when they see
```

```
   "Connection: close" in the request. The effect will be that keep-alive will
   never be used. A workaround consists in enabling "option
   http-pretend-keepalive".

   At the moment, logs will not indicate whether requests came from the same
   session or not. The accept date reported in the logs corresponds to the end
   of the previous request, and the request time corresponds to the time spent
   waiting for a new request. The keep-alive request time is still bound to the
   timeout defined by "timeout http-keep-alive" or "timeout http-request" if
   not set.

   This option may be set both in a frontend and in a backend. It is enabled if
   at least one of the frontend or backend holding a connection has it enabled.
   It disables and replaces any previous "option httpclose", "option forceclose",
   "option http-tunnel" or "option http-keep-alive". Please check section 4
   ("Proxies") to see how this option combines with others when frontend and
   backend options differ.

   If this option has been enabled in a "defaults" section, it can be disabled
   in a specific instance by prepending the "no" keyword before it.

   See also : "option forceclose", "option http-pretend-keepalive",
              "option httpclose", "option http-keep-alive", and
              "1.1. The HTTP transaction model".


option http-tunnel
no option http-tunnel
  Disable or enable HTTP connection processing after first transaction
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments : none

  By default HAProxy operates in keep-alive mode with regards to persistent
  connections: for each connection it processes each request and response, and
  leaves the connection idle on both sides between the end of a response and
  the start of a new request. This mode may be changed by several options such
  as "option http-server-close", "option forceclose", "option httpclose" or
  "option http-tunnel".

  Option "http-tunnel" disables any HTTP processing past the first request and
  the first response. This is the mode which was used by default in versions
  1.0 to 1.5-dev21. It is the mode with the lowest processing overhead, which
  is normally not needed anymore unless in very specific cases such as when
  using an in-house protocol that looks like HTTP but is not compatible, or
  just to log one request per client in order to reduce log size. Note that
  everything which works at the HTTP level, including header parsing/addition,
  cookie processing or content switching will only work for the first request
  and will be ignored after the first response.

  If this option has been enabled in a "defaults" section, it can be disabled
  in a specific instance by prepending the "no" keyword before it.

  See also : "option forceclose", "option http-server-close",
             "option httpclose", "option http-keep-alive", and
             "1.1. The HTTP transaction model".


option http-use-proxy-header
no option http-use-proxy-header
  Make use of non-standard Proxy-Connection header instead of Connection
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments : none

```



```
  While RFC2616 explicitly states that HTTP/1.1 agents must use the
  Connection header to indicate their wish of persistent or non-persistent
  connections, both browsers and proxies ignore this header for proxied
  connections and make use of the undocumented, non-standard Proxy-Connection
  header instead. The issue begins when trying to put a load balancer between
  browsers and such proxies, because there will be a difference between what
  haproxy understands and what the client and the proxy agree on.

  By setting this option in a frontend, haproxy can automatically switch to use
  that non-standard header if it sees proxied requests. A proxied request is
  defined here as one where the URI begins with neither a '/' nor a '*'. The
  choice of header only affects requests passing through proxies making use of
  one of the "httpclose", "forceclose" and "http-server-close" options. Note
  that this option can only be specified in a frontend and will affect the
  request along its whole life.

  Also, when this option is set, a request which requires authentication will
  automatically switch to use proxy authentication headers if it is itself a
  proxied request. That makes it possible to check or enforce authentication in
  front of an existing proxy.

  This option should normally never be used, except in front of a proxy.

  See also : "option httpclose", "option forceclose" and "option
             http-server-close".


option httpchk
option httpchk <uri>
option httpchk <method> <uri>
option httpchk <method> <uri> <version>
  Enable HTTP protocol to check on the servers health
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <method>  is the optional HTTP method used with the requests. When not set,
              the "OPTIONS" method is used, as it generally requires low server
              processing and is easy to filter out from the logs. Any method
              may be used, though it is not recommended to invent non-standard
              ones.

    <uri>     is the URI referenced in the HTTP requests. It defaults to " / "
              which is accessible by default on almost any server, but may be
              changed to any other URI. Query strings are permitted.

    <version> is the optional HTTP version string. It defaults to "HTTP/1.0"
              but some servers might behave incorrectly in HTTP 1.0, so turning
              it to HTTP/1.1 may sometimes help. Note that the Host field is
              mandatory in HTTP/1.1, and as a trick, it is possible to pass it
              after "\r\n" following the version string.

  By default, server health checks only consist in trying to establish a TCP
  connection. When "option httpchk" is specified, a complete HTTP request is
  sent once the TCP connection is established, and responses 2xx and 3xx are
  considered valid, while all other ones indicate a server failure, including
  the lack of any response.

  The port and interval are specified in the server configuration.

  This option does not necessarily require an HTTP backend, it also works with
  plain TCP backends. This is particularly useful to check simple scripts bound
  to some dedicated ports using the inetd daemon.

  Examples :
      # Relay HTTPS traffic to Apache instance and check service availability
```

```
4421        # using HTTP request "OPTIONS * HTTP/1.1" on port 80.
4422        backend https_relay
4423            mode tcp
4424            option httpchk OPTIONS * HTTP/1.1\r\nHost:\ www
4425            server apache1 192.168.1.1:443 check port 80
4426
4427    See also : "option ssl-hello-chk", "option smtpchk", "option mysql-check",
4428               "option pgsql-check", "http-check" and the "check", "port" and
4429               "inter" server options.
4430
4431
4432  option httpclose
4433  no option httpclose
4434    Enable or disable passive HTTP connection closing
4435    May be used in sections :   defaults | frontend | listen | backend
4436                                   yes   |    yes   |   yes  |   yes
4437    Arguments : none
4438
4439    By default HAProxy operates in keep-alive mode with regards to persistent
4440    connections: for each connection it processes each request and response, and
4441    leaves the connection idle on both sides between the end of a response and
4442    the start of a new request. This mode may be changed by several options such
4443    as "option http-server-close", "option forceclose", "option httpclose" or
4444    "option http-tunnel".
4445
4446    If "option httpclose" is set, HAProxy will work in HTTP tunnel mode and check
4447    if a "Connection: close" header is already set in each direction, and will
4448    add one if missing. Each end should react to this by actively closing the TCP
4449    connection after each transfer, thus resulting in a switch to the HTTP close
4450    mode. Any "Connection" header different from "close" will also be removed.
4451    Note that this option is deprecated since what it does is very cheap but not
4452    reliable. Using "option http-server-close" or "option forceclose" is strongly
4453    recommended instead.
4454
4455    It seldom happens that some servers incorrectly ignore this header and do not
4456    close the connection even though they reply "Connection: close". For this
4457    reason, they are not compatible with older HTTP 1.0 browsers. If this happens
4458    it is possible to use the "option forceclose" which actively closes the
4459    request connection once the server responds. Option "forceclose" also
4460    releases the server connection earlier because it does not have to wait for
4461    the client to acknowledge it.
4462
4463    This option may be set both in a frontend and in a backend. It is enabled if
4464    at least one of the frontend or backend holding a connection has it enabled.
4465    It disables and replaces any previous "option http-server-close",
4466    "option forceclose", "option http-keep-alive" or "option http-tunnel". Please
4467    check section 4 ("Proxies") to see how this option combines with others when
4468    frontend and backend options differ.
4469
4470    If this option has been enabled in a "defaults" section, it can be disabled
4471    in a specific instance by prepending the "no" keyword before it.
4472
4473    See also : "option forceclose", "option http-server-close" and
4474               "1.1. The HTTP transaction model".
4475
4476
4477  option httplog [ clf ]
4478    Enable logging of HTTP request, session state and timers
4479    May be used in sections :   defaults | frontend | listen | backend
4480                                   yes   |    yes   |   yes  |   yes
4481    Arguments :
4482      clf       if the "clf" argument is added, then the output format will be
4483                the CLF format instead of HAProxy's default HTTP format. You can
4484                use this when you need to feed HAProxy's logs through a specific
4485                log analyser which only support the CLF format and which is not
```



```
4486                extensible.
4487
4488    By default, the log output format is very poor, as it only contains the
4489    source and destination addresses, and the instance name. By specifying
4490    "option httplog", each log line turns into a much richer format including,
4491    but not limited to, the HTTP request, the connection timers, the session
4492    status, the connections numbers, the captured headers and cookies, the
4493    frontend, backend and server name, and of course the source address and
4494    ports.
4495
4496    This option may be set either in the frontend or the backend.
4497
4498    Specifying only "option httplog" will automatically clear the 'clf' mode
4499    if it was set by default.
4500
4501    See also :  section 8 about logging.
4502
4503
4504  option http_proxy
4505  no option http_proxy
4506    Enable or disable plain HTTP proxy mode
4507    May be used in sections :   defaults | frontend | listen | backend
4508                                   yes   |    yes   |   yes  |   yes
4509    Arguments : none
4510
4511    It sometimes happens that people need a pure HTTP proxy which understands
4512    basic proxy requests without caching nor any fancy feature. In this case,
4513    it may be worth setting up an HAProxy instance with the "option http_proxy"
4514    set. In this mode, no server is declared, and the connection is forwarded to
4515    the IP address and port found in the URL after the "http://" scheme.
4516
4517    No host address resolution is performed, so this only works when pure IP
4518    addresses are passed. Since this option's usage perimeter is rather limited,
4519    it will probably be used only by experts who know they need exactly it. Last,
4520    if the clients are susceptible of sending keep-alive requests, it will be
4521    needed to add "option httpclose" to ensure that all requests will correctly
4522    be analyzed.
4523
4524    If this option has been enabled in a "defaults" section, it can be disabled
4525    in a specific instance by prepending the "no" keyword before it.
4526
4527    Example :
4528      # this backend understands HTTP proxy requests and forwards them directly.
4529      backend direct_forward
4530          option httpclose
4531          option http_proxy
4532
4533    See also : "option httpclose"
4534
4535
4536  option independent-streams
4537  no option independent-streams
4538    Enable or disable independent timeout processing for both directions
4539    May be used in sections :   defaults | frontend | listen | backend
4540                                   yes   |    yes   |   yes  |  yes
4541    Arguments : none
4542
4543    By default, when data is sent over a socket, both the write timeout and the
4544    read timeout for that socket are refreshed, because we consider that there is
4545    activity on that socket, and we have no other means of guessing if we should
4546    receive data or not.
4547
4548    While this default behaviour is desirable for almost all applications, there
4549    exists a situation where it is desirable to disable it, and only refresh the
4550    read timeout if there are incoming data. This happens on sessions with large
```

```
   timeouts and low amounts of exchanged data such as telnet session. If the
   server suddenly disappears, the output data accumulates in the system's
   socket buffers, both timeouts are correctly refreshed, and there is no way
   to know the server does not receive them, so we don't timeout. However, when
   the underlying protocol always echoes sent data, it would be enough by itself
   to detect the issue using the read timeout. Note that this problem does not
   happen with more verbose protocols because data won't accumulate long in the
   socket buffers.

   When this option is set on the frontend, it will disable read timeout updates
   on data sent to the client. There probably is little use of this case. When
   the option is set on the backend, it will disable read timeout updates on
   data sent to the server. Doing so will typically break large HTTP posts from
   slow lines, so use it with caution.

   Note: older versions used to call this setting "option independent-streams"
         with a spelling mistake. This spelling is still supported but
         deprecated.

   See also : "timeout client", "timeout server" and "timeout tunnel"


option ldap-check
  Use LDAPv3 health checks for server testing
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments : none

  It is possible to test that the server correctly talks LDAPv3 instead of just
  testing that it accepts the TCP connection. When this option is set, an
  LDAPv3 anonymous simple bind message is sent to the server, and the response
  is analyzed to find an LDAPv3 bind response message.

  The server is considered valid only when the LDAP response contains success
  resultCode (http://tools.ietf.org/html/rfc4511#section-4.1.9).

  Logging of bind requests is server dependent see your documentation how to
  configure it.

  Example :
        option ldap-check

  See also : "option httpchk"


option log-health-checks
no option log-health-checks
  Enable or disable logging of health checks status updates
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |  yes
  Arguments : none

  By default, failed health check are logged if server is UP and successful
  health checks are logged if server is DOWN, so the amount of additional
  information is limited.

  When this option is enabled, any change of the health check status or to
  the server's health will be logged, so that it becomes possible to know
  that a server was failing occasional checks before crashing, or exactly when
  it failed to respond a valid HTTP status, then when the port started to
  reject connections, then when the server stopped responding at all.

  Note that status changes not caused by health checks (eg: enable/disable on
  the CLI) are intentionally not logged by this option.

```



```
  See also: "option httpchk", "option ldap-check", "option mysql-check",
            "option pgsql-check", "option redis-check", "option smtpchk",
            "option tcp-check", "log" and section 8 about logging.


option log-separate-errors
no option log-separate-errors
  Change log level for non-completely successful connections
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments : none

  Sometimes looking for errors in logs is not easy. This option makes haproxy
  raise the level of logs containing potentially interesting information such
  as errors, timeouts, retries, redispatches, or HTTP status codes 5xx. The
  level changes from "info" to "err". This makes it possible to log them
  separately to a different file with most syslog daemons. Be careful not to
  remove them from the original file, otherwise you would lose ordering which
  provides very important information.

  Using this option, large sites dealing with several thousand connections per
  second may log normal traffic to a rotating buffer and only archive smaller
  error logs.

  See also : "log", "dontlognull", "dontlog-normal" and section 8 about
             logging.


option logasap
no option logasap
  Enable or disable early logging of HTTP requests
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments : none

  By default, HTTP requests are logged upon termination so that the total
  transfer time and the number of bytes appear in the logs. When large objects
  are being transferred, it may take a while before the request appears in the
  logs. Using "option logasap", the request gets logged as soon as the server
  sends the complete headers. The only missing information in the logs will be
  the total number of bytes which will indicate everything except the amount
  of data transferred, and the total time which will not take the transfer
  time into account. In such a situation, it's a good practice to capture the
  "Content-Length" response header so that the logs at least indicate how many
  bytes are expected to be transferred.

  Examples :
      listen http_proxy 0.0.0.0:80
          mode http
          option httplog
          option logasap
          log 192.168.2.200 local3

    >>> Feb  6 12:14:14 localhost \
          haproxy[14389]: 10.0.1.2:33317 [06/Feb/2009:12:14:14.655] http-in \
          static/srv1 9/10/7/14/+30 200 +243 - - ---- 3/1/1/1/0 1/0 \
          "GET /image.iso HTTP/1.0"

  See also : "option httplog", "capture response header", and section 8 about
             logging.


option mysql-check [ user <username> [ post-41 ] ]
  Use MySQL health checks for server testing
  May be used in sections :   defaults | frontend | listen | backend
```

```
                                  yes   |    no    |   yes  |   yes
  Arguments :
    <username> This is the username which will be used when connecting to MySQL
               server.
    post-41    Send post v4.1 client compatible checks

  If you specify a username, the check consists of sending two MySQL packet,
  one Client Authentication packet, and one QUIT packet, to correctly close
  MySQL session. We then parse the MySQL Handshake Initialisation packet and/or
  Error packet. It is a basic but useful test which does not produce error nor
  aborted connect on the server. However, it requires adding an authorization
  in the MySQL table, like this :

      USE mysql;
      INSERT INTO user (Host,User) values ('<ip_of_haproxy>','<username>');
      FLUSH PRIVILEGES;

  If you don't specify a username (it is deprecated and not recommended), the
  check only consists in parsing the Mysql Handshake Initialisation packet or
  Error packet, we don't send anything in this mode. It was reported that it
  can generate lockout if check is too frequent and/or if there is not enough
  traffic. In fact, you need in this case to check MySQL "max_connect_errors"
  value as if a connection is established successfully within fewer than MySQL
  "max_connect_errors" attempts after a previous connection was interrupted,
  the error count for the host is cleared to zero. If HAProxy's server get
  blocked, the "FLUSH HOSTS" statement is the only way to unblock it.

  Remember that this does not check database presence nor database consistency.
  To do this, you can use an external check with xinetd for example.

  The check requires MySQL >=3.22, for older version, please use TCP check.

  Most often, an incoming MySQL server needs to see the client's IP address for
  various purposes, including IP privilege matching and connection logging.
  When possible, it is often wise to masquerade the client's IP address when
  connecting to the server using the "usesrc" argument of the "source" keyword,
  which requires the cttproxy feature to be compiled in, and the MySQL server
  to route the client via the machine hosting haproxy.

  See also: "option httpchk"


option nolinger
no option nolinger
  Enable or disable immediate session resource cleaning after close
  May be used in sections:    defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
  Arguments : none

  When clients or servers abort connections in a dirty way (eg: they are
  physically disconnected), the session timeouts triggers and the session is
  closed. But it will remain in FIN_WAIT1 state for some time in the system,
  using some resources and possibly limiting the ability to establish newer
  connections.

  When this happens, it is possible to activate "option nolinger" which forces
  the system to immediately remove any socket's pending data on close. Thus,
  the session is instantly purged from the system's tables. This usually has
  side effects such as increased number of TCP resets due to old retransmits
  getting immediately rejected. Some firewalls may sometimes complain about
  this too.

  For this reason, it is not recommended to use this option when not absolutely
  needed. You know that you need it when you have thousands of FIN_WAIT1
  sessions on your system (TIME_WAIT ones do not count).
```



```

  This option may be used both on frontends and backends, depending on the side
  where it is required. Use it on the frontend for clients, and on the backend
  for servers.

  If this option has been enabled in a "defaults" section, it can be disabled
  in a specific instance by prepending the "no" keyword before it.


option originalto [ except <network> ] [ header <name> ]
  Enable insertion of the X-Original-To header to requests sent to servers
  May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
  Arguments :
    <network> is an optional argument used to disable this option for sources
              matching <network>
    <name>    an optional argument to specify a different "X-Original-To"
              header name.

  Since HAProxy can work in transparent mode, every request from a client can
  be redirected to the proxy and HAProxy itself can proxy every request to a
  complex SQUID environment and the destination host from SO_ORIGINAL_DST will
  be lost. This is annoying when you want access rules based on destination ip
  addresses. To solve this problem, a new HTTP header "X-Original-To" may be
  added by HAProxy to all requests sent to the server. This header contains a
  value representing the original destination IP address. Since this must be
  configured to always use the last occurrence of this header only. Note that
  only the last occurrence of the header must be used, since it is really
  possible that the client has already brought one.

  The keyword "header" may be used to supply a different header name to replace
  the default "X-Original-To". This can be useful where you might already
  have a "X-Original-To" header from a different application, and you need
  preserve it. Also if your backend server doesn't use the "X-Original-To"
  header and requires different one.

  Sometimes, a same HAProxy instance may be shared between a direct client
  access and a reverse-proxy access (for instance when an SSL reverse-proxy is
  used to decrypt HTTPS traffic). It is possible to disable the addition of the
  header for a known source address or network by adding the "except" keyword
  followed by the network address. In this case, any source IP matching the
  network will not cause an addition of this header. Most common uses are with
  private networks or 127.0.0.1.

  This option may be specified either in the frontend or in the backend. If at
  least one of them uses it, the header will be added. Note that the backend's
  setting of the header subargument takes precedence over the frontend's if
  both are defined.

  Examples :
    # Original Destination address
    frontend www
        mode http
        option originalto except 127.0.0.1

    # Those servers want the IP Address in X-Client-Dst
    backend www
        mode http
        option originalto header X-Client-Dst

  See also : "option httpclose", "option http-server-close",
             "option forceclose"


option persist
```

```
no option persist
  Enable or disable forced persistence on down servers
  May be used in sections:    defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments : none

  When an HTTP request reaches a backend with a cookie which references a dead
  server, by default it is redispatched to another server. It is possible to
  force the request to be sent to the dead server first using "option persist"
  if absolutely needed. A common use case is when servers are under extreme
  load and spend their time flapping. In this case, the users would still be
  directed to the server they opened the session on, in the hope they would be
  correctly served. It is recommended to use "option redispatch" in conjunction
  with this option so that in the event it would not be possible to connect to
  the server at all (server definitely dead), the client would finally be
  redirected to another valid server.

  If this option has been enabled in a "defaults" section, it can be disabled
  in a specific instance by prepending the "no" keyword before it.

  See also : "option redispatch", "retries", "force-persist"


option pgsql-check [ user <username> ]
  Use PostgreSQL health checks for server testing
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <username> This is the username which will be used when connecting to
               PostgreSQL server.

  The check sends a PostgreSQL StartupMessage and waits for either
  Authentication request or ErrorResponse message. It is a basic but useful
  test which does not produce error nor aborted connect on the server.
  This check is identical with the "mysql-check".

  See also: "option httpchk"


option prefer-last-server
no option prefer-last-server
  Allow multiple load balanced requests to remain on the same server
  May be used in sections:    defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments : none

  When the load balancing algorithm in use is not deterministic, and a previous
  request was sent to a server to which haproxy still holds a connection, it is
  sometimes desirable that subsequent requests on a same session go to the same
  server as much as possible. Note that this is different from persistence, as
  we only indicate a preference which haproxy tries to apply without any form
  of warranty. The real use is for keep-alive connections sent to servers. When
  this option is used, haproxy will try to reuse the same connection that is
  attached to the server instead of rebalancing to another server, causing a
  close of the connection. This can make sense for static file servers. It does
  not make much sense to use this in combination with hashing algorithms. Note,
  haproxy already automatically tries to stick to a server which sends a 401 or
  to a proxy which sends a 407 (authentication required). This is mandatory for
  use with the broken NTLM authentication challenge, and significantly helps in
  troubleshooting some faulty applications. Option prefer-last-server might be
  desirable in these environments as well, to avoid redistributing the traffic
  after every other response.

  If this option has been enabled in a "defaults" section, it can be disabled
  in a specific instance by prepending the "no" keyword before it.
```



```
  See also: "option http-keep-alive"


option redispatch
no option redispatch
  Enable or disable session redistribution in case of connection failure
  May be used in sections:    defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments : none

  In HTTP mode, if a server designated by a cookie is down, clients may
  definitely stick to it because they cannot flush the cookie, so they will not
  be able to access the service anymore.

  Specifying "option redispatch" will allow the proxy to break their
  persistence and redistribute them to a working server.

  It also allows to retry last connection to another server in case of multiple
  connection failures. Of course, it requires having "retries" set to a nonzero
  value.

  This form is the preferred form, which replaces both the "redispatch" and
  "redisp" keywords.

  If this option has been enabled in a "defaults" section, it can be disabled
  in a specific instance by prepending the "no" keyword before it.

  See also : "redispatch", "retries", "force-persist"


option redis-check
  Use redis health checks for server testing
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments : none

  It is possible to test that the server correctly talks REDIS protocol instead
  of just testing that it accepts the TCP connection. When this option is set,
  a PING redis command is sent to the server, and the response is analyzed to
  find the "+PONG" response message.

  Example :
        option redis-check

  See also : "option httpchk"


option smtpchk
option smtpchk <hello> <domain>
  Use SMTP health checks for server testing
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <hello>   is an optional argument. It is the "hello" command to use. It can
              be either "HELO" (for SMTP) or "EHLO" (for ESTMP). All other
              values will be turned into the default command ("HELO").

    <domain>  is the domain name to present to the server. It may only be
              specified (and is mandatory) if the hello command has been
              specified. By default, "localhost" is used.

  When "option smtpchk" is set, the health checks will consist in TCP
  connections followed by an SMTP command. By default, this command is
  "HELO localhost". The server's return code is analyzed and only return codes
```

```
   starting with a "2" will be considered as valid. All other responses,
   including a lack of response will constitute an error and will indicate a
   dead server.

   This test is meant to be used with SMTP servers or relays. Depending on the
   request, it is possible that some servers do not log each connection attempt,
   so you may want to experiment to improve the behaviour. Using telnet on port
   25 is often easier than adjusting the configuration.

   Most often, an incoming SMTP server needs to see the client's IP address for
   various purposes, including spam filtering, anti-spoofing and logging. When
   possible, it is often wise to masquerade the client's IP address when
   connecting to the server using the "usesrc" argument of the "source" keyword,
   which requires the cttproxy feature to be compiled in.

   Example :
         option smtpchk HELO mydomain.org

   See also : "option httpchk", "source"


option socket-stats
no option socket-stats

   Enable or disable collecting & providing separate statistics for each socket.
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   no

   Arguments : none


option splice-auto
no option splice-auto
   Enable or disable automatic kernel acceleration on sockets in both directions
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
   Arguments : none

   When this option is enabled either on a frontend or on a backend, haproxy
   will automatically evaluate the opportunity to use kernel tcp splicing to
   forward data between the client and the server, in either direction. Haproxy
   uses heuristics to estimate if kernel splicing might improve performance or
   not. Both directions are handled independently. Note that the heuristics used
   are not much aggressive in order to limit excessive use of splicing. This
   option requires splicing to be enabled at compile time, and may be globally
   disabled with the global option "nosplice". Since splice uses pipes, using it
   requires that there are enough spare pipes.

   Important note: kernel-based TCP splicing is a Linux-specific feature which
   first appeared in kernel 2.6.25. It offers kernel-based acceleration to
   transfer data between sockets without copying these data to user-space, thus
   providing noticeable performance gains and CPU cycles savings. Since many
   early implementations are buggy, corrupt data and/or are inefficient, this
   feature is not enabled by default, and it should be used with extreme care.
   While it is not possible to detect the correctness of an implementation,
   2.6.29 is the first version offering a properly working implementation. In
   case of doubt, splicing may be globally disabled using the global "nosplice"
   keyword.

   Example :
         option splice-auto

   If this option has been enabled in a "defaults" section, it can be disabled
   in a specific instance by prepending the "no" keyword before it.

```



```
   See also : "option splice-request", "option splice-response", and global
              options "nosplice" and "maxpipes"


option splice-request
no option splice-request
   Enable or disable automatic kernel acceleration on sockets for requests
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
   Arguments : none

   When this option is enabled either on a frontend or on a backend, haproxy
   will use kernel tcp splicing whenever possible to forward data going from
   the client to the server. It might still use the recv/send scheme if there
   are no spare pipes left. This option requires splicing to be enabled at
   compile time, and may be globally disabled with the global option "nosplice".
   Since splice uses pipes, using it requires that there are enough spare pipes.

   Important note: see "option splice-auto" for usage limitations.

   Example :
         option splice-request

   If this option has been enabled in a "defaults" section, it can be disabled
   in a specific instance by prepending the "no" keyword before it.

   See also : "option splice-auto", "option splice-response", and global options
              "nosplice" and "maxpipes"


option splice-response
no option splice-response
   Enable or disable automatic kernel acceleration on sockets for responses
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    yes   |   yes  |   yes
   Arguments : none

   When this option is enabled either on a frontend or on a backend, haproxy
   will use kernel tcp splicing whenever possible to forward data going from
   the server to the client. It might still use the recv/send scheme if there
   are no spare pipes left. This option requires splicing to be enabled at
   compile time, and may be globally disabled with the global option "nosplice".
   Since splice uses pipes, using it requires that there are enough spare pipes.

   Important note: see "option splice-auto" for usage limitations.

   Example :
         option splice-response

   If this option has been enabled in a "defaults" section, it can be disabled
   in a specific instance by prepending the "no" keyword before it.

   See also : "option splice-auto", "option splice-request", and global options
              "nosplice" and "maxpipes"


option srvtcpka
no option srvtcpka
   Enable or disable the sending of TCP keepalive packets on the server side
   May be used in sections :   defaults | frontend | listen | backend
                                  yes   |    no    |   yes  |   yes
   Arguments : none

   When there is a firewall or any session-aware component between a client and
   a server, and when the protocol involves very long sessions with long idle
```

```
   periods (eg: remote desktops), there is a risk that one of the intermediate
   components decides to expire a session which has remained idle for too long.

   Enabling socket-level TCP keep-alives makes the system regularly send packets
   to the other end of the connection, leaving it active. The delay between
   keep-alive probes is controlled by the system only and depends both on the
   operating system and its tuning parameters.

   It is important to understand that keep-alive packets are neither emitted nor
   received at the application level. It is only the network stacks which sees
   them. For this reason, even if one side of the proxy already uses keep-alives
   to maintain its connection alive, those keep-alive packets will not be
   forwarded to the other side of the proxy.

   Please note that this has nothing to do with HTTP keep-alive.

   Using option "srvtcpka" enables the emission of TCP keep-alive probes on the
   server side of a connection, which should help when session expirations are
   noticed between HAProxy and a server.

   If this option has been enabled in a "defaults" section, it can be disabled
   in a specific instance by prepending the "no" keyword before it.

   See also : "option clitcpka", "option tcpka"


option ssl-hello-chk
  Use SSLv3 client hello health checks for server testing
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments : none

  When some SSL-based protocols are relayed in TCP mode through HAProxy, it is
  possible to test that the server correctly talks SSL instead of just testing
  that it accepts the TCP connection. When "option ssl-hello-chk" is set, pure
  SSLv3 client hello messages are sent once the connection is established to
  the server, and the response is analyzed to find an SSL server hello message.
  The server is considered valid only when the response contains this server
  hello message.

  All servers tested till there correctly reply to SSLv3 client hello messages,
  and most servers tested do not even log the requests containing only hello
  messages, which is appreciable.

  Note that this check works even when SSL support was not built into haproxy
  because it forges the SSL message. When SSL support is available, it is best
  to use native SSL health checks instead of this one.

  See also: "option httpchk", "check-ssl"


option tcp-check
  Perform health checks using tcp-check send/expect sequences
  May be used in sections:   defaults | frontend | listen | backend
                               yes    |    no    |   yes  |   yes

  This health check method is intended to be combined with "tcp-check" command
  lists in order to support send/expect types of health check sequences.

  TCP checks currently support 4 modes of operations :
    - no "tcp-check" directive : the health check only consists in a connection
      attempt, which remains the default mode.

    - "tcp-check send" or "tcp-check send-binary" only is mentioned : this is
      used to send a string along with a connection opening. With some
```



```
      protocols, it helps sending a "QUIT" message for example that prevents
      the server from logging a connection error for each health check. The
      check result will still be based on the ability to open the connection
      only.

    - "tcp-check expect" only is mentioned : this is used to test a banner.
      The connection is opened and haproxy waits for the server to present some
      contents which must validate some rules. The check result will be based
      on the matching between the contents and the rules. This is suited for
      POP, IMAP, SMTP, FTP, SSH, TELNET.

    - both "tcp-check send" and "tcp-check expect" are mentioned : this is
      used to test a hello-type protocol. Haproxy sends a message, the server
      responds and its response is analysed. the check result will be based on
      the matching between the response contents and the rules. This is often
      suited for protocols which require a binding or a request/response model.
      LDAP, MySQL, Redis and SSL are example of such protocols, though they
      already all have their dedicated checks with a deeper understanding of
      the respective protocols.
      In this mode, many questions may be sent and many answers may be
      analysed.

  Examples :
        # perform a POP check (analyse only server's banner)
        option tcp-check
        tcp-check expect string +OK\ POP3\ ready

        # perform an IMAP check (analyse only server's banner)
        option tcp-check
        tcp-check expect string *\ OK\ IMAP4\ ready

        # look for the redis master server after ensuring it speaks well
        # redis protocol, then it exits properly.
        # (send a command then analyse the response 3 times)
        option tcp-check
        tcp-check send PING\r\n
        tcp-check expect +PONG
        tcp-check send info\ replication\r\n
        tcp-check expect string role:master
        tcp-check send QUIT\r\n
        tcp-check expect string +OK

        forge a HTTP request, then analyse the response
        (send many headers before analyzing)
        option tcp-check
        tcp-check send HEAD\ /\ HTTP/1.1\r\n
        tcp-check send Host:\ www.mydomain.com\r\n
        tcp-check send User-Agent:\ HAProxy\ tcpcheck\r\n
        tcp-check send \r\n
        tcp-check expect rstring HTTP/1\..\ (2..|3..)


  See also : "tcp-check expect", "tcp-check send"


option tcp-smart-accept
no option tcp-smart-accept
  Enable or disable the saving of one ACK packet during the accept sequence
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |    no
  Arguments : none

  When an HTTP connection request comes in, the system acknowledges it on
  behalf of HAProxy, then the client immediately sends its request, and the
  system acknowledges it too while it is notifying HAProxy about the new
```

connection. HAProxy then reads the request and responds. This means that we have one TCP ACK sent by the system for nothing, because the request could very well be acknowledged by HAProxy when it sends its response.

For this reason, in HTTP mode, HAProxy automatically asks the system to avoid sending this useless ACK on platforms which support it (currently at least Linux). It must not cause any problem, because the system will send it anyway after 40 ms if the response takes more time than expected to come.

During complex network debugging sessions, it may be desirable to disable this optimization because delayed ACKs can make troubleshooting more complex when trying to identify where packets are delayed. It is then possible to fall back to normal behaviour by specifying "no option tcp-smart-accept".

It is also possible to force it for non-HTTP proxies by simply specifying "option tcp-smart-accept". For instance, it can make sense with some services such as SMTP where the server speaks first.

It is recommended to avoid forcing this option in a defaults section. In case of doubt, consider setting it back to automatic values by prepending the "default" keyword before it, or disabling it using the "no" keyword.

See also : "option tcp-smart-connect"

option tcp-smart-connect
no option tcp-smart-connect
  Enable or disable the saving of one ACK packet during the connect sequence
  May be used in sections :   defaults | frontend | listen | backend
                                 yes | no | yes | yes
  Arguments : none

  On certain systems (at least Linux), HAProxy can ask the kernel not to immediately send an empty ACK upon a connection request, but to directly send the buffer request instead. This saves one packet on the network and thus boosts performance. It can also be useful for some servers, because they immediately get the request along with the incoming connection.

  This feature is enabled when "option tcp-smart-connect" is set in a backend. It is not enabled by default because it makes network troubleshooting more complex.

  It only makes sense to enable it with protocols where the client speaks first such as HTTP. In other situations, if there is no data to send in place of the ACK, a normal ACK is sent.

  If this option has been enabled in a "defaults" section, it can be disabled in a specific instance by prepending the "no" keyword before it.

  See also : "option tcp-smart-accept"

option tcpka
  Enable or disable the sending of TCP keepalive packets on both sides
  May be used in sections :   defaults | frontend | listen | backend
                                 yes | yes | yes | yes
  Arguments : none

  When there is a firewall or any session-aware component between a client and a server, and when the protocol involves very long sessions with long idle periods (eg: remote desktops), there is a risk that one of the intermediate components decides to expire a session which has remained idle for too long.

  Enabling socket-level TCP keep-alives makes the system regularly send packets to the other end of the connection, leaving it active. The delay between

  keep-alive probes is controlled by the system only and depends both on the operating system and its tuning parameters.

  It is important to understand that keep-alive packets are neither emitted nor received at the application level. It is only the network stacks which sees them. For this reason, even if one side of the proxy already uses keep-alives to maintain its connection alive, those keep-alive packets will not be forwarded to the other side of the proxy.

  Please note that this has nothing to do with HTTP keep-alive.

  Using option "tcpka" enables the emission of TCP keep-alive probes on both the client and server sides of a connection. Note that this is meaningful only in "defaults" or "listen" sections. If this option is used in a frontend, only the client side will get keep-alives, and if this option is used in a backend, only the server side will get keep-alives. For this reason, it is strongly recommended to explicitly use "option clitcpka" and "option srvtcpka" when the configuration is split between frontends and backends.

  See also : "option clitcpka", "option srvtcpka"

option tcplog
  Enable advanced logging of TCP connections with session state and timers
  May be used in sections :   defaults | frontend | listen | backend
                                 yes | yes | yes | yes
  Arguments : none

  By default, the log output format is very poor, as it only contains the source and destination addresses, and the instance name. By specifying "option tcplog", each log line turns into a much richer format including, but not limited to, the connection timers, the session status, the connections numbers, the frontend, backend and server name, and of course the source address and ports. This option is useful for pure TCP proxies in order to find which of the client or server disconnects or times out. For normal HTTP proxies, it's better to use "option httplog" which is even more complete.

  This option may be set either in the frontend or the backend.

  See also : "option httplog", and section 8 about logging.

option transparent
no option transparent
  Enable client-side transparent proxying
  May be used in sections :   defaults | frontend | listen | backend
                                 yes | no | yes | yes
  Arguments : none

  This option was introduced in order to provide layer 7 persistence to layer 3 load balancers. The idea is to use the OS's ability to redirect an incoming connection for a remote address to a local process (here HAProxy), and let this process know what address was initially requested. When this option is used, sessions without cookies will be forwarded to the original destination IP address of the incoming request (which should match that of another equipment), while requests with cookies will still be forwarded to the appropriate server.

  Note that contrary to a common belief, this option does NOT make HAProxy present the client's IP to the server when establishing the connection.

  See also: the "usesrc" argument of the "source" keyword, and the
            "transparent" option of the "bind" keyword.

```
5331
5332  persist rdp-cookie
5333  persist rdp-cookie(<name>)
5334    Enable RDP cookie-based persistence
5335    May be used in sections :   defaults | frontend | listen | backend
5336                                   yes   |    no    |   yes  |   yes
5337    Arguments :
5338      <name>    is the optional name of the RDP cookie to check. If omitted, the
5339                default cookie name "msts" will be used. There currently is no
5340                valid reason to change this name.
5341
5342    This statement enables persistence based on an RDP cookie. The RDP cookie
5343    contains all information required to find the server in the list of known
5344    servers. So when this option is set in the backend, the request is analysed
5345    and if an RDP cookie is found, it is decoded. If it matches a known server
5346    which is still UP (or if "option persist" is set), then the connection is
5347    forwarded to this server.
5348
5349    Note that this only makes sense in a TCP backend, but for this to work, the
5350    frontend must have waited long enough to ensure that an RDP cookie is present
5351    in the request buffer. This is the same requirement as with the "rdp-cookie"
5352    load-balancing method. Thus it is highly recommended to put all statements in
5353    a single "listen" section.
5354
5355    Also, it is important to understand that the terminal server will emit this
5356    RDP cookie only if it is configured for "token redirection mode", which means
5357    that the "IP address redirection" option is disabled.
5358
5359    Example :
5360          listen tse-farm
5361              bind :3389
5362              # wait up to 5s for an RDP cookie in the request
5363              tcp-request inspect-delay 5s
5364              tcp-request content accept if RDP_COOKIE
5365              # apply RDP cookie persistence
5366              persist rdp-cookie
5367              # if server is unknown, let's balance on the same cookie.
5368              # alternatively, "balance leastconn" may be useful too.
5369              balance rdp-cookie
5370              server srv1 1.1.1.1:3389
5371              server srv2 1.1.1.2:3389
5372
5373    See also : "balance rdp-cookie", "tcp-request", the "req_rdp_cookie" ACL and
5374    the rdp_cookie pattern fetch function.
5375
5376
5377  rate-limit sessions <rate>
5378    Set a limit on the number of new sessions accepted per second on a frontend
5379    May be used in sections :   defaults | frontend | listen | backend
5380                                   yes   |    yes   |   yes  |   no
5381    Arguments :
5382      <rate>    The <rate> parameter is an integer designating the maximum number
5383                of new sessions per second to accept on the frontend.
5384
5385    When the frontend reaches the specified number of new sessions per second, it
5386    stops accepting new connections until the rate drops below the limit again.
5387    During this time, the pending sessions will be kept in the socket's backlog
5388    (in system buffers) and haproxy will not even be aware that sessions are
5389    pending. When applying very low limit on a highly loaded service, it may make
5390    sense to increase the socket's backlog using the "backlog" keyword.
5391
5392    This feature is particularly efficient at blocking connection-based attacks
5393    or service abuse on fragile servers. Since the session rate is measured every
5394    millisecond, it is extremely accurate. Also, the limit applies immediately,
5395    no delay is needed at all to detect the threshold.
```



```
5396
5397    Example : limit the connection rate on SMTP to 10 per second max
5398          listen smtp
5399              mode tcp
5400              bind :25
5401              rate-limit sessions 10
5402              server 127.0.0.1:1025
5403
5404    Note : when the maximum rate is reached, the frontend's status is not changed
5405           but its sockets appear as "WAITING" in the statistics if the
5406           "socket-stats" option is enabled.
5407
5408    See also : the "backlog" keyword and the "fe_sess_rate" ACL criterion.
5409
5410
5411  redirect location <loc> [code <code>] <option> [{if | unless} <condition>]
5412  redirect prefix   <pfx> [code <code>] <option> [{if | unless} <condition>]
5413  redirect scheme   <sch> [code <code>] <option> [{if | unless} <condition>]
5414    Return an HTTP redirection if/unless a condition is matched
5415    May be used in sections :   defaults | frontend | listen | backend
5416                                   no    |    yes   |   yes  |   yes
5417
5418    If/unless the condition is matched, the HTTP request will lead to a redirect
5419    response. If no condition is specified, the redirect applies unconditionally.
5420
5421    Arguments :
5422      <loc>     With "redirect location", the exact value in <loc> is placed into
5423                the HTTP "Location" header. When used in an "http-request" rule,
5424                <loc> value follows the log-format rules and can include some
5425                dynamic values (see Custom Log Format in section 8.2.4).
5426
5427      <pfx>     With "redirect prefix", the "Location" header is built from the
5428                concatenation of <pfx> and the complete URI path, including the
5429                query string, unless the "drop-query" option is specified (see
5430                below). As a special case, if <pfx> equals exactly "/", then
5431                nothing is inserted before the original URI. It allows one to
5432                redirect to the same URL (for instance, to insert a cookie). When
5433                used in an "http-request" rule, <pfx> value follows the log-format
5434                rules and can include some dynamic values (see Custom Log Format
5435                in section 8.2.4).
5436
5437      <sch>     With "redirect scheme", then the "Location" header is built by
5438                concatenating <sch> with "://" then the first occurrence of the
5439                "Host" header, and then the URI path, including the query string
5440                unless the "drop-query" option is specified (see below). If no
5441                path is found or if the path is "*", then "/" is used instead. If
5442                no "Host" header is found, then an empty host component will be
5443                returned, which most recent browsers interpret as redirecting to
5444                the same host. This directive is mostly used to redirect HTTP to
5445                HTTPS. When used in an "http-request" rule, <sch> value follows
5446                the log-format rules and can include some dynamic values (see
5447                Custom Log Format in section 8.2.4).
5448
5449      <code>    The code is optional. It indicates which type of HTTP redirection
5450                is desired. Only codes 301, 302, 303, 307 and 308 are supported,
5451                with 302 used by default if no code is specified. 301 means
5452                "Moved permanently", and a browser may cache the Location. 302
5453                means "Moved temporarily" and means that the browser should not
5454                cache the redirection. 303 is equivalent to 302 except that the
5455                browser will fetch the location with a GET method. 307 is just
5456                like 302 but makes it clear that the same method must be reused.
5457                Likewise, 308 replaces 301 if the same method must be used.
5458
5459      <option>  There are several options which can be specified to adjust the
5460                expected behaviour of a redirection :
```

```
5461
5462        - "drop-query"
5463          When this keyword is used in a prefix-based redirection, then the
5464          location will be set without any possible query-string, which is useful
5465          for directing users to a non-secure page for instance. It has no effect
5466          with a location-type redirect.
5467
5468        - "append-slash"
5469          This keyword may be used in conjunction with "drop-query" to redirect
5470          users who use a URL not ending with a '/' to the same one with the '/'.
5471          It can be useful to ensure that search engines will only see one URL.
5472          For this, a return code 301 is preferred.
5473
5474        - "set-cookie NAME[=value]"
5475          A "Set-Cookie" header will be added with NAME (and optionally "=value")
5476          to the response. This is sometimes used to indicate that a user has
5477          been seen, for instance to protect against some types of DoS. No other
5478          cookie option is added, so the cookie will be a session cookie. Note
5479          that for a browser, a sole cookie name without an equal sign is
5480          different from a cookie with an equal sign.
5481
5482        - "clear-cookie NAME[=]"
5483          A "Set-Cookie" header will be added with NAME (and optionally "="), but
5484          with the "Max-Age" attribute set to zero. This will tell the browser to
5485          delete this cookie. It is useful for instance on logout pages. It is
5486          important to note that clearing the cookie "NAME" will not remove a
5487          cookie set with "NAME=value". You have to clear the cookie "NAME=" for
5488          that, because the browser makes the difference.
5489
5490    Example: move the login URL only to HTTPS.
5491          acl clear      dst_port  80
5492          acl secure     dst_port  8080
5493          acl login_page url_beg   /login
5494          acl logout     url_beg   /logout
5495          acl uid_given  url_reg   /login?userid=[^&]+
5496          acl cookie_set hdr_sub(cookie) SEEN=1
5497
5498          redirect prefix   https://mysite.com set-cookie SEEN=1 if !cookie_set
5499          redirect prefix   https://mysite.com           if login_page !secure
5500          redirect prefix   http://mysite.com drop-query if login_page !uid_given
5501          redirect location http://mysite.com/           if !login_page secure
5502          redirect location / clear-cookie USERID=       if logout
5503
5504    Example: send redirects for request for articles without a '/'.
5505          acl missing_slash path_reg ^/article/[^/]*$
5506          redirect code 301 prefix / drop-query append-slash if missing_slash
5507
5508    Example: redirect all HTTP traffic to HTTPS when SSL is handled by haproxy.
5509          redirect scheme https if !{ ssl_fc }
5510
5511    Example: append 'www.' prefix in front of all hosts not having it
5512          http-request redirect code 301 location www.%[hdr(host)]%[req.uri] \
5513            unless { hdr_beg(host) -i www }
5514
5515    See section 7 about ACL usage.
5516
5517
5518  redisp (deprecated)
5519  redispatch (deprecated)
5520    Enable or disable session redistribution in case of connection failure
5521    May be used in sections:    defaults | frontend | listen | backend
5522                                   yes   |    no    |   yes  |   yes
5523    Arguments : none
5524
5525    In HTTP mode, if a server designated by a cookie is down, clients may
```



```
5526    definitely stick to it because they cannot flush the cookie, so they will not
5527    be able to access the service anymore.
5528
5529    Specifying "redispatch" will allow the proxy to break their persistence and
5530    redistribute them to a working server.
5531
5532    It also allows to retry last connection to another server in case of multiple
5533    connection failures. Of course, it requires having "retries" set to a nonzero
5534    value.
5535
5536    This form is deprecated, do not use it in any new configuration, use the new
5537    "option redispatch" instead.
5538
5539    See also : "option redispatch"
5540
5541
5542  reqadd  <string> [{if | unless} <cond>]
5543    Add a header at the end of the HTTP request
5544    May be used in sections :   defaults | frontend | listen | backend
5545                                   no    |    yes   |   yes  |   yes
5546    Arguments :
5547      <string>  is the complete line to be added. Any space or known delimiter
5548                must be escaped using a backslash ('\'). Please refer to section
5549                6 about HTTP header manipulation for more information.
5550
5551      <cond>    is an optional matching condition built from ACLs. It makes it
5552                possible to ignore this rule when other conditions are not met.
5553
5554    A new line consisting in <string> followed by a line feed will be added after
5555    the last header of an HTTP request.
5556
5557    Header transformations only apply to traffic which passes through HAProxy,
5558    and not to traffic generated by HAProxy, such as health-checks or error
5559    responses.
5560
5561    Example : add "X-Proto: SSL" to requests coming via port 81
5562       acl is-ssl  dst_port       81
5563       reqadd      X-Proto:\ SSL  if is-ssl
5564
5565    See also: "rspadd", section 6 about HTTP header manipulation, and section 7
5566              about ACLs.
5567
5568
5569  reqallow  <search> [{if | unless} <cond>]
5570  reqiallow <search> [{if | unless} <cond>] (ignore case)
5571    Definitely allow an HTTP request if a line matches a regular expression
5572    May be used in sections :   defaults | frontend | listen | backend
5573                                   no    |    yes   |   yes  |   yes
5574    Arguments :
5575      <search>  is the regular expression applied to HTTP headers and to the
5576                request line. This is an extended regular expression. Parenthesis
5577                grouping is supported and no preliminary backslash is required.
5578                Any space or known delimiter must be escaped using a backslash
5579                ('\'). The pattern applies to a full line at a time. The
5580                "reqallow" keyword strictly matches case while "reqiallow"
5581                ignores case.
5582
5583      <cond>    is an optional matching condition built from ACLs. It makes it
5584                possible to ignore this rule when other conditions are not met.
5585
5586    A request containing any line which matches extended regular expression
5587    <search> will mark the request as allowed, even if any later test would
5588    result in a deny. The test applies both to the request line and to request
5589    headers. Keep in mind that URLs in request line are case-sensitive while
5590    header names are not.
```

```
5591
5592    It is easier, faster and more powerful to use ACLs to write access policies.
5593    Reqdeny, reqallow and reqpass should be avoided in new designs.
5594
5595    Example :
5596       # allow www.* but refuse *.local
5597       reqiallow ^Host:\ www\.
5598       reqideny  ^Host:\ .*\.local
5599
5600    See also: "reqdeny", "block", section 6 about HTTP header manipulation, and
5601              section 7 about ACLs.
5602
5603
5604  reqdel  <search> [{if | unless} <cond>]
5605  reqidel <search> [{if | unless} <cond>]  (ignore case)
5606    Delete all headers matching a regular expression in an HTTP request
5607    May be used in sections :   defaults | frontend | listen | backend
5608                                   no    |    yes   |   yes  |   yes
5609    Arguments :
5610      <search>  is the regular expression applied to HTTP headers and to the
5611                request line. This is an extended regular expression. Parenthesis
5612                grouping is supported and no preliminary backslash is required.
5613                Any space or known delimiter must be escaped using a backslash
5614                ('\'). The pattern applies to a full line at a time. The "reqdel"
5615                keyword strictly matches case while "reqidel" ignores case.
5616
5617      <cond>    is an optional matching condition built from ACLs. It makes it
5618                possible to ignore this rule when other conditions are not met.
5619
5620    Any header line matching extended regular expression <search> in the request
5621    will be completely deleted. Most common use of this is to remove unwanted
5622    and/or dangerous headers or cookies from a request before passing it to the
5623    next servers.
5624
5625    Header transformations only apply to traffic which passes through HAProxy,
5626    and not to traffic generated by HAProxy, such as health-checks or error
5627    responses. Keep in mind that header names are not case-sensitive.
5628
5629    Example :
5630       # remove X-Forwarded-For header and SERVER cookie
5631       reqidel ^X-Forwarded-For:.*
5632       reqidel ^Cookie:.*SERVER=
5633
5634    See also: "reqadd", "reqrep", "rspdel", section 6 about HTTP header
5635              manipulation, and section 7 about ACLs.
5636
5637
5638  reqdeny  <search> [{if | unless} <cond>]
5639  reqideny <search> [{if | unless} <cond>]  (ignore case)
5640    Deny an HTTP request if a line matches a regular expression
5641    May be used in sections :   defaults | frontend | listen | backend
5642                                   no    |    yes   |   yes  |   yes
5643    Arguments :
5644      <search>  is the regular expression applied to HTTP headers and to the
5645                request line. This is an extended regular expression. Parenthesis
5646                grouping is supported and no preliminary backslash is required.
5647                Any space or known delimiter must be escaped using a backslash
5648                ('\'). The pattern applies to a full line at a time. The
5649                "reqdeny" keyword strictly matches case while "reqideny" ignores
5650                case.
5651
5652      <cond>    is an optional matching condition built from ACLs. It makes it
5653                possible to ignore this rule when other conditions are not met.
5654
5655    A request containing any line which matches extended regular expression
```



```
5656    <search> will mark the request as denied, even if any later test would
5657    result in an allow. The test applies both to the request line and to request
5658    headers. Keep in mind that URLs in request line are case-sensitive while
5659    header names are not.
5660
5661    A denied request will generate an "HTTP 403 forbidden" response once the
5662    complete request has been parsed. This is consistent with what is practiced
5663    using ACLs.
5664
5665    It is easier, faster and more powerful to use ACLs to write access policies.
5666    Reqdeny, reqallow and reqpass should be avoided in new designs.
5667
5668    Example :
5669       # refuse *.local, then allow www.*
5670       reqideny  ^Host:\ .*\.local
5671       reqiallow ^Host:\ www\.
5672
5673    See also: "reqallow", "rspdeny", "block", section 6 about HTTP header
5674              manipulation, and section 7 about ACLs.
5675
5676
5677  reqpass  <search> [{if | unless} <cond>]
5678  reqipass <search> [{if | unless} <cond>]  (ignore case)
5679    Ignore any HTTP request line matching a regular expression in next rules
5680    May be used in sections :   defaults | frontend | listen | backend
5681                                   no    |    yes   |   yes  |   yes
5682    Arguments :
5683      <search>  is the regular expression applied to HTTP headers and to the
5684                request line. This is an extended regular expression. Parenthesis
5685                grouping is supported and no preliminary backslash is required.
5686                Any space or known delimiter must be escaped using a backslash
5687                ('\'). The pattern applies to a full line at a time. The
5688                "reqpass" keyword strictly matches case while "reqipass" ignores
5689                case.
5690
5691      <cond>    is an optional matching condition built from ACLs. It makes it
5692                possible to ignore this rule when other conditions are not met.
5693
5694    A request containing any line which matches extended regular expression
5695    <search> will skip next rules, without assigning any deny or allow verdict.
5696    The test applies both to the request line and to request headers. Keep in
5697    mind that URLs in request line are case-sensitive while header names are not.
5698
5699    It is easier, faster and more powerful to use ACLs to write access policies.
5700    Reqdeny, reqallow and reqpass should be avoided in new designs.
5701
5702    Example :
5703       # refuse *.local, then allow www.*, but ignore "www.private.local"
5704       reqipass  ^Host:\ www.private\.local
5705       reqideny  ^Host:\ .*\.local
5706       reqiallow ^Host:\ www\.
5707
5708    See also: "reqallow", "reqdeny", "block", section 6 about HTTP header
5709              manipulation, and section 7 about ACLs.
5710
5711
5712  reqrep  <search> <string> [{if | unless} <cond>]
5713  reqirep <search> <string> [{if | unless} <cond>]   (ignore case)
5714    Replace a regular expression with a string in an HTTP request line
5715    May be used in sections :   defaults | frontend | listen | backend
5716                                   no    |    yes   |   yes  |   yes
5717    Arguments :
5718      <search>  is the regular expression applied to HTTP headers and to the
5719                request line. This is an extended regular expression. Parenthesis
5720                grouping is supported and no preliminary backslash is required.
```

```
               Any space or known delimiter must be escaped using a backslash
               ('\'). The pattern applies to a full line at a time. The "reqrep"
               keyword strictly matches case while "reqirep" ignores case.

     <string>  is the complete line to be added. Any space or known delimiter
               must be escaped using a backslash ('\'). References to matched
               pattern groups are possible using the common \N form, with N
               being a single digit between 0 and 9. Please refer to section
               6 about HTTP header manipulation for more information.

     <cond>    is an optional matching condition built from ACLs. It makes it
               possible to ignore this rule when other conditions are not met.

  Any line matching extended regular expression <search> in the request (both
  the request line and header lines) will be completely replaced with <string>.
  Most common use of this is to rewrite URLs or domain names in "Host" headers.

  Header transformations only apply to traffic which passes through HAProxy,
  and not to traffic generated by HAProxy, such as health-checks or error
  responses. Note that for increased readability, it is suggested to add enough
  spaces between the request and the response. Keep in mind that URLs in
  request line are case-sensitive while header names are not.

  Example :
     # replace "/static/" with "/" at the beginning of any request path.
     reqrep ^([^\ :]*)\ /static/(.*)     \1\ /\2
     # replace "www.mydomain.com" with "www" in the host name.
     reqirep ^Host:\ www.mydomain.com   Host:\ www

  See also: "reqadd", "reqdel", "rsprep", "tune.bufsize", section 6 about
            HTTP header manipulation, and section 7 about ACLs.


reqtarpit  <search> [{if | unless} <cond>]
reqitarpit <search> [{if | unless} <cond>]  (ignore case)
  Tarpit an HTTP request containing a line matching a regular expression
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes
  Arguments :
    <search>  is the regular expression applied to HTTP headers and to the
              request line. This is an extended regular expression. Parenthesis
              grouping is supported and no preliminary backslash is required.
              Any space or known delimiter must be escaped using a backslash
              ('\'). The pattern applies to a full line at a time. The
              "reqtarpit" keyword strictly matches case while "reqitarpit"
              ignores case.

    <cond>    is an optional matching condition built from ACLs. It makes it
              possible to ignore this rule when other conditions are not met.

  A request containing any line which matches extended regular expression
  <search> will be tarpitted, which means that it will connect to nowhere, will
  be kept open for a pre-defined time, then will return an HTTP error 500 so
  that the attacker does not suspect it has been tarpitted. The status 500 will
  be reported in the logs, but the completion flags will indicate "PT". The
  delay is defined by "timeout tarpit", or "timeout connect" if the former is
  not set.

  The goal of the tarpit is to slow down robots attacking servers with
  identifiable requests. Many robots limit their outgoing number of connections
  and stay connected waiting for a reply which can take several minutes to
  come. Depending on the environment and attack, it may be particularly
  efficient at reducing the load on the network and firewalls.

  Examples :
```



```
     # ignore user-agents reporting any flavour of "Mozilla" or "MSIE", but
     # block all others.
     reqipass   ^User-Agent:\.*(Mozilla|MSIE)
     reqitarpit ^User-Agent:

     # block bad guys
     acl badguys src 10.1.0.3 172.16.13.20/28
     reqitarpit . if badguys

  See also: "reqallow", "reqdeny", "reqpass", section 6 about HTTP header
            manipulation, and section 7 about ACLs.


retries <value>
  Set the number of retries to perform on a server after a connection failure
  May be used in sections:    defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments :
    <value>   is the number of times a connection attempt should be retried on
              a server when a connection either is refused or times out. The
              default value is 3.

  It is important to understand that this value applies to the number of
  connection attempts, not full requests. When a connection has effectively
  been established to a server, there will be no more retry.

  In order to avoid immediate reconnections to a server which is restarting,
  a turn-around timer of 1 second is applied before a retry occurs.

  When "option redispatch" is set, the last retry may be performed on another
  server even if a cookie references a different server.

  See also : "option redispatch"


rspadd <string> [{if | unless} <cond>]
  Add a header at the end of the HTTP response
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes
  Arguments :
    <string>  is the complete line to be added. Any space or known delimiter
              must be escaped using a backslash ('\'). Please refer to section
              6 about HTTP header manipulation for more information.

    <cond>    is an optional matching condition built from ACLs. It makes it
              possible to ignore this rule when other conditions are not met.

  A new line consisting in <string> followed by a line feed will be added after
  the last header of an HTTP response.

  Header transformations only apply to traffic which passes through HAProxy,
  and not to traffic generated by HAProxy, such as health-checks or error
  responses.

  See also: "reqadd", section 6 about HTTP header manipulation, and section 7
            about ACLs.


rspdel  <search> [{if | unless} <cond>]
rspidel <search> [{if | unless} <cond>]  (ignore case)
  Delete all headers matching a regular expression in an HTTP response
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes
  Arguments :
    <search>  is the regular expression applied to HTTP headers and to the
```

```
               response line. This is an extended regular expression, so
               parenthesis grouping is supported and no preliminary backslash
               is required. Any space or known delimiter must be escaped using
               a backslash ('\'). The pattern applies to a full line at a time.
               The "rspdel" keyword strictly matches case while "rspidel"
               ignores case.

    <cond>     is an optional matching condition built from ACLs. It makes it
               possible to ignore this rule when other conditions are not met.

  Any header line matching extended regular expression <search> in the response
  will be completely deleted. Most common use of this is to remove unwanted
  and/or sensitive headers or cookies from a response before passing it to the
  client.

  Header transformations only apply to traffic which passes through HAProxy,
  and not to traffic generated by HAProxy, such as health-checks or error
  responses. Keep in mind that header names are not case-sensitive.

  Example :
     # remove the Server header from responses
     rspidel ^Server:.*

  See also: "rspadd", "rsprep", "reqdel", section 6 about HTTP header
            manipulation, and section 7 about ACLs.


rspdeny  <search> [{if | unless} <cond>]
rspideny <search> [{if | unless} <cond>]  (ignore case)
  Block an HTTP response if a line matches a regular expression
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes
  Arguments :
    <search>  is the regular expression applied to HTTP headers and to the
              response line. This is an extended regular expression, so
              parenthesis grouping is supported and no preliminary backslash
              is required. Any space or known delimiter must be escaped using
              a backslash ('\'). The pattern applies to a full line at a time.
              The "rspdeny" keyword strictly matches case while "rspideny"
              ignores case.

    <cond>    is an optional matching condition built from ACLs. It makes it
              possible to ignore this rule when other conditions are not met.

  A response containing any line which matches extended regular expression
  <search> will mark the request as denied. The test applies both to the
  response line and to response headers. Keep in mind that header names are not
  case-sensitive.

  Main use of this keyword is to prevent sensitive information leak and to
  block the response before it reaches the client. If a response is denied, it
  will be replaced with an HTTP 502 error so that the client never retrieves
  any sensitive data.

  It is easier, faster and more powerful to use ACLs to write access policies.
  Rspdeny should be avoided in new designs.

  Example :
     # Ensure that no content type matching ms-word will leak
     rspideny  ^Content-type:\.*/ms-word

  See also: "reqdeny", "acl", "block", section 6 about HTTP header manipulation
            and section 7 about ACLs.
```



```
rsprep  <search> <string> [{if | unless} <cond>]
rspirep <search> <string> [{if | unless} <cond>]  (ignore case)
  Replace a regular expression with a string in an HTTP response line
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   yes
  Arguments :
    <search>  is the regular expression applied to HTTP headers and to the
              response line. This is an extended regular expression, so
              parenthesis grouping is supported and no preliminary backslash
              is required. Any space or known delimiter must be escaped using
              a backslash ('\'). The pattern applies to a full line at a time.
              The "rsprep" keyword strictly matches case while "rspirep"
              ignores case.

    <string>  is the complete line to be added. Any space or known delimiter
              must be escaped using a backslash ('\'). References to matched
              pattern groups are possible using the common \N form, with N
              being a single digit between 0 and 9. Please refer to section
              6 about HTTP header manipulation for more information.

    <cond>    is an optional matching condition built from ACLs. It makes it
              possible to ignore this rule when other conditions are not met.

  Any line matching extended regular expression <search> in the response (both
  the response line and header lines) will be completely replaced with
  <string>. Most common use of this is to rewrite Location headers.

  Header transformations only apply to traffic which passes through HAProxy,
  and not to traffic generated by HAProxy, such as health-checks or error
  responses. Note that for increased readability, it is suggested to add enough
  spaces between the request and the response. Keep in mind that header names
  are not case-sensitive.

  Example :
     # replace "Location: 127.0.0.1:8080" with "Location: www.mydomain.com"
     rspirep ^Location:\ 127.0.0.1:8080    Location:\ www.mydomain.com

  See also: "rspadd", "rspdel", "reqrep", section 6 about HTTP header
            manipulation, and section 7 about ACLs.


server <name> <address>[:[port]] [param*]
  Declare a server in a backend
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    no    |   yes  |   yes
  Arguments :
    <name>    is the internal name assigned to this server. This name will
              appear in logs and alerts.  If "http-send-name-header" is
              set, it will be added to the request header sent to the server.

    <address> is the IPv4 or IPv6 address of the server. Alternatively, a
              resolvable hostname is supported, but this name will be resolved
              during start-up. Address "0.0.0.0" or "*" has a special meaning.
              It indicates that the connection will be forwarded to the same IP
              address as the one from the client connection. This is useful in
              transparent proxy architectures where the client's connection is
              intercepted and haproxy must forward to the original destination
              address. This is more or less what the "transparent" keyword does
              except that with a server it's possible to limit concurrency and
              to report statistics. Optionally, an address family prefix may be
              used before the address to force the family regardless of the
              address format, which can be useful to specify a path to a unix
              socket with no slash ('/'). Currently supported prefixes are :
                    - 'ipv4@'  -> address is always IPv4
                    - 'ipv6@'  -> address is always IPv6
```

```
                    - 'unix@'  -> address is a path to a local unix socket
                    - 'abns@'  -> address is in abstract namespace (Linux only)
                Any part of the address string may reference any number of
                environment variables by preceding their name with a dollar
                sign ('$') and optionally enclosing them with braces ('{}'),
                similarly to what is done in Bourne shell.

      <port>    is an optional port specification. If set, all connections will
                be sent to this port. If unset, the same port the client
                connected to will be used. The port may also be prefixed by a "+"
                or a "-". In this case, the server's port will be determined by
                adding this value to the client's port.

      <param*>  is a list of parameters for this server. The "server" keywords
                accepts an important number of options and has a complete section
                dedicated to it. Please refer to section 5 for more details.

    Examples :
          server first  10.1.1.1:1080 cookie first  check inter 1000
          server second 10.1.1.2:1080 cookie second check inter 1000
          server transp ipv4@
          server backup ${SRV_BACKUP}:1080 backup
          server www1_dc1 ${LAN_DC1}.101:80
          server www1_dc2 ${LAN_DC2}.101:80

    Note: regarding Linux's abstract namespace sockets, HAProxy uses the whole
          sun_path length is used for the address length. Some other programs
          such as socat use the string length only by default. Pass the option
          ",unix-tightsocklen=0" to any abstract socket definition in socat to
          make it compatible with HAProxy's.

    See also: "default-server", "http-send-name-header" and section 5 about
              server options


  source <addr>[:<port>] [usesrc { <addr2>[:<port2>] | client | clientip } ]
  source <addr>[:<port>] [usesrc { <addr2>[:<port2>] | hdr_ip(<hdr>[,<occ>]) } ]
  source <addr>[:<port>] [interface <name>]
    Set the source address for outgoing connections
    May be used in sections :   defaults | frontend | listen | backend
                                   yes   |    no    |   yes  |   yes
    Arguments :
      <addr>    is the IPv4 address HAProxy will bind to before connecting to a
                server. This address is also used as a source for health checks.

                The default value of 0.0.0.0 means that the system will select
                the most appropriate address to reach its destination. Optionally
                an address family prefix may be used before the address to force
                the family regardless of the address format, which can be useful
                to specify a path to a unix socket with no slash ('/'). Currently
                supported prefixes are :
                  - 'ipv4@' -> address is always IPv4
                  - 'ipv6@' -> address is always IPv6
                  - 'unix@' -> address is a path to a local unix socket
                  - 'abns@' -> address is in abstract namespace (Linux only)
                Any part of the address string may reference any number of
                environment variables by preceding their name with a dollar
                sign ('$') and optionally enclosing them with braces ('{}'),
                similarly to what is done in Bourne shell.

      <port>    is an optional port. It is normally not needed but may be useful
                in some very specific contexts. The default value of zero means
                the system will select a free port. Note that port ranges are not
                supported in the backend. If you want to force port ranges, you
                have to specify them on each "server" line.
```



```

      <addr2>   is the IP address to present to the server when connections are
                forwarded in full transparent proxy mode. This is currently only
                supported on some patched Linux kernels. When this address is
                specified, clients connecting to the server will be presented
                with this address, while health checks will still use the address
                <addr>.

      <port2>   is the optional port to present to the server when connections
                are forwarded in full transparent proxy mode (see <addr2> above).
                The default value of zero means the system will select a free
                port.

      <hdr>     is the name of a HTTP header in which to fetch the IP to bind to.
                This is the name of a comma-separated header list which can
                contain multiple IP addresses. By default, the last occurrence is
                used. This is designed to work with the X-Forwarded-For header
                and to automatically bind to the client's IP address as seen
                by previous proxy, typically Stunnel. In order to use another
                occurrence from the last one, please see the <occ> parameter
                below. When the header (or occurrence) is not found, no binding
                is performed so that the proxy's default IP address is used. Also
                keep in mind that the header name is case insensitive, as for any
                HTTP header.

      <occ>     is the occurrence number of a value to be used in a multi-value
                header. This is to be used in conjunction with "hdr_ip(<hdr>)",
                in order to specify which occurrence to use for the source IP
                address. Positive values indicate a position from the first
                occurrence, 1 being the first one. Negative values indicate
                positions relative to the last one, -1 being the last one. This
                is helpful for situations where an X-Forwarded-For header is set
                at the entry point of an infrastructure and must be used several
                proxy layers away. When this value is not specified, -1 is
                assumed. Passing a zero here disables the feature.

      <name>    is an optional interface name to which to bind to for outgoing
                traffic. On systems supporting this features (currently, only
                Linux), this allows one to bind all traffic to the server to
                this interface even if it is not the one the system would select
                based on routing tables. This should be used with extreme care.
                Note that using this option requires root privileges.

    The "source" keyword is useful in complex environments where a specific
    address only is allowed to connect to the servers. It may be needed when a
    private address must be used through a public gateway for instance, and it is
    known that the system cannot determine the adequate source address by itself.

    An extension which is available on certain patched Linux kernels may be used
    through the "usesrc" optional keyword. It makes it possible to connect to the
    servers with an IP address which does not belong to the system itself. This
    is called "full transparent proxy mode". For this to work, the destination
    servers have to route their traffic back to this address through the machine
    running HAProxy, and IP forwarding must generally be enabled on this machine.

    In this "full transparent proxy" mode, it is possible to force a specific IP
    address to be presented to the servers. This is not much used in fact. A more
    common use is to tell HAProxy to present the client's IP address. For this,
    there are two methods :

      - present the client's IP and port addresses. This is the most transparent
        mode, but it can cause problems when IP connection tracking is enabled on
        the machine, because a same connection may be seen twice with different
        states. However, this solution presents the huge advantage of not
        limiting the system to the 64k outgoing address+port couples, because all
```

```
        of the client ranges may be used.

      - present only the client's IP address and select a spare port. This
        solution is still quite elegant but slightly less transparent (downstream
        firewalls logs will not match upstream's). It also presents the downside
        of limiting the number of concurrent connections to the usual 64k ports.
        However, since the upstream and downstream ports are different, local IP
        connection tracking on the machine will not be upset by the reuse of the
        same session.

    Note that depending on the transparent proxy technology used, it may be
    required to force the source address. In fact, cttproxy version 2 requires an
    IP address in <addr> above, and does not support setting of "0.0.0.0" as the
    IP address because it creates NAT entries which much match the exact outgoing
    address. Tproxy version 4 and some other kernel patches which work in pure
    forwarding mode generally will not have this limitation.

    This option sets the default source for all servers in the backend. It may
    also be specified in a "defaults" section. Finer source address specification
    is possible at the server level using the "source" server option. Refer to
    section 5 for more information.

    In order to work, "usesrc" requires root privileges.

    Examples :
          backend private
              # Connect to the servers using our 192.168.1.200 source address
              source 192.168.1.200

          backend transparent_ssl1
              # Connect to the SSL farm from the client's source address
              source 192.168.1.200 usesrc clientip

          backend transparent_ssl2
              # Connect to the SSL farm from the client's source address and port
              # not recommended if IP conntrack is present on the local machine.
              source 192.168.1.200 usesrc client

          backend transparent_ssl3
              # Connect to the SSL farm from the client's source address. It
              # is more conntrack-friendly.
              source 192.168.1.200 usesrc clientip

          backend transparent_smtp
              # Connect to the SMTP farm from the client's source address/port
              # with Tproxy version 4.
              source 0.0.0.0 usesrc clientip

          backend transparent_http
              # Connect to the servers using the client's IP as seen by previous
              # proxy.
              source 0.0.0.0 usesrc hdr_ip(x-forwarded-for,-1)

    See also : the "source" server option in section 5, the Tproxy patches for
               the Linux kernel on www.balabit.com, the "bind" keyword.


  srvtimeout <timeout> (deprecated)
    Set the maximum inactivity time on the server side.
    May be used in sections :   defaults | frontend | listen | backend
                                   yes   |    no    |   yes  |   yes
    Arguments :
      <timeout> is the timeout value specified in milliseconds by default, but
                can be in any other unit if the number is suffixed by the unit,
                as explained at the top of this document.
```



```

    The inactivity timeout applies when the server is expected to acknowledge or
    send data. In HTTP mode, this timeout is particularly important to consider
    during the first phase of the server's response, when it has to send the
    headers, as it directly represents the server's processing time for the
    request. To find out what value to put there, it's often good to start with
    what would be considered as unacceptable response times, then check the logs
    to observe the response time distribution, and adjust the value accordingly.

    The value is specified in milliseconds by default, but can be in any other
    unit if the number is suffixed by the unit, as specified at the top of this
    document. In TCP mode (and to a lesser extent, in HTTP mode), it is highly
    recommended that the client timeout remains equal to the server timeout in
    order to avoid complex situations to debug. Whatever the expected server
    response times, it is a good practice to cover at least one or several TCP
    packet losses by specifying timeouts that are slightly above multiples of 3
    seconds (eg: 4 or 5 seconds minimum).

    This parameter is specific to backends, but can be specified once for all in
    "defaults" sections. This is in fact one of the easiest solutions not to
    forget about it. An unspecified timeout results in an infinite timeout, which
    is not recommended. Such a usage is accepted and works but reports a warning
    during startup because it may results in accumulation of expired sessions in
    the system if the system's timeouts are not configured either.

    This parameter is provided for compatibility but is currently deprecated.
    Please use "timeout server" instead.

    See also : "timeout server", "timeout tunnel", "timeout client" and
               "clitimeout".


  stats admin { if | unless } <cond>
    Enable statistics admin level if/unless a condition is matched
    May be used in sections :   defaults | frontend | listen | backend
                                   no    |    yes   |   yes  |   yes

    This statement enables the statistics admin level if/unless a condition is
    matched.

    The admin level allows to enable/disable servers from the web interface. By
    default, statistics page is read-only for security reasons.

    Note : Consider not using this feature in multi-process mode (nbproc > 1)
           unless you know what you do : memory is not shared between the
           processes, which can result in random behaviours.

    Currently, the POST request is limited to the buffer size minus the reserved
    buffer space, which means that if the list of servers is too long, the
    request won't be processed. It is recommended to alter few servers at a
    time.

    Example :
      # statistics admin level only for localhost
      backend stats_localhost
          stats enable
          stats admin if LOCALHOST

    Example :
      # statistics admin level always enabled because of the authentication
      backend stats_auth
          stats enable
          stats auth  admin:AdMiN123
          stats admin if TRUE

```

```
   Example :
     # statistics admin level depends on the authenticated user
     userlist stats-auth
         group admin    users admin
         user  admin    insecure-password AdMiN123
         group readonly users haproxy
         user  haproxy  insecure-password haproxy

     backend stats_auth
         stats enable
         acl AUTH       http_auth(stats-auth)
         acl AUTH_ADMIN http_auth_group(stats-auth) admin
         stats http-request auth unless AUTH
         stats admin if AUTH_ADMIN

   See also : "stats enable", "stats auth", "stats http-request", "nbproc",
              "bind-process", section 3.4 about userlists and section 7 about
              ACL usage.


stats auth <user>:<passwd>
  Enable statistics with authentication and grant access to an account
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments :
    <user>    is a user name to grant access to

    <passwd>  is the cleartext password associated to this user

  This statement enables statistics with default settings, and restricts access
  to declared users only. It may be repeated as many times as necessary to
  allow as many users as desired. When a user tries to access the statistics
  without a valid account, a "401 Forbidden" response will be returned so that
  the browser asks the user to provide a valid user and password. The real
  which will be returned to the browser is configurable using "stats realm".

  Since the authentication method is HTTP Basic Authentication, the passwords
  circulate in cleartext on the network. Thus, it was decided that the
  configuration file would also use cleartext passwords to remind the users
  that those ones should not be sensitive and not shared with any other account.

  It is also possible to reduce the scope of the proxies which appear in the
  report using "stats scope".

  Though this statement alone is enough to enable statistics reporting, it is
  recommended to set all other settings in order to avoid relying on default
  unobvious parameters.

  Example :
    # public access (limited to this backend only)
    backend public_www
        server srv1 192.168.0.1:80
        stats enable
        stats hide-version
        stats scope   .
        stats uri     /admin?stats
        stats realm   Haproxy\ Statistics
        stats auth    admin1:AdMiN123
        stats auth    admin2:AdMiN321

    # internal monitoring access (unlimited)
    backend private_monitoring
        stats enable
        stats uri     /admin?stats
        stats refresh 5s
```



```

  See also : "stats enable", "stats realm", "stats scope", "stats uri"


stats enable
  Enable statistics reporting with default settings
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments : none

  This statement enables statistics reporting with default settings defined
  at build time. Unless stated otherwise, these settings are used :
    - stats uri   : /haproxy?stats
    - stats realm : "HAProxy Statistics"
    - stats auth  : no authentication
    - stats scope : no restriction

  Though this statement alone is enough to enable statistics reporting, it is
  recommended to set all other settings in order to avoid relying on default
  unobvious parameters.

  Example :
    # public access (limited to this backend only)
    backend public_www
        server srv1 192.168.0.1:80
        stats enable
        stats hide-version
        stats scope   .
        stats uri     /admin?stats
        stats realm   Haproxy\ Statistics
        stats auth    admin1:AdMiN123
        stats auth    admin2:AdMiN321

    # internal monitoring access (unlimited)
    backend private_monitoring
        stats enable
        stats uri     /admin?stats
        stats refresh 5s

  See also : "stats auth", "stats realm", "stats uri"


stats hide-version
  Enable statistics and hide HAProxy version reporting
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments : none

  By default, the stats page reports some useful status information along with
  the statistics. Among them is HAProxy's version. However, it is generally
  considered dangerous to report precise version to anyone, as it can help them
  target known weaknesses with specific attacks. The "stats hide-version"
  statement removes the version from the statistics report. This is recommended
  for public sites or any site with a weak login/password.

  Though this statement alone is enough to enable statistics reporting, it is
  recommended to set all other settings in order to avoid relying on default
  unobvious parameters.

  Example :
    # public access (limited to this backend only)
    backend public_www
        server srv1 192.168.0.1:80
        stats enable
        stats hide-version
```

```
6371          stats scope   .
6372          stats uri     /admin?stats
6373          stats realm   Haproxy\ Statistics
6374          stats auth    admin1:AdMiN123
6375          stats auth    admin2:AdMiN321
6376
6377      # internal monitoring access (unlimited)
6378      backend private_monitoring
6379          stats enable
6380          stats uri     /admin?stats
6381          stats refresh 5s
6382
6383    See also : "stats auth", "stats enable", "stats realm", "stats uri"
6384
6385
6386  stats http-request { allow | deny | auth [realm <realm>] }
6387              [ { if | unless } <condition> ]
6388    Access control for statistics
6389
6390    May be used in sections:   defaults | frontend | listen | backend
6391                                  no    |    no    |   yes  |   yes
6392
6393    As "http-request", these set of options allow to fine control access to
6394    statistics. Each option may be followed by if/unless and acl.
6395    First option with matched condition (or option without condition) is final.
6396    For "deny" a 403 error will be returned, for "allow" normal processing is
6397    performed, for "auth" a 401/407 error code is returned so the client
6398    should be asked to enter a username and password.
6399
6400    There is no fixed limit to the number of http-request statements per
6401    instance.
6402
6403    See also : "http-request", section 3.4 about userlists and section 7
6404               about ACL usage.
6405
6406
6407  stats realm <realm>
6408    Enable statistics and set authentication realm
6409    May be used in sections :   defaults | frontend | listen | backend
6410                                   yes   |    yes   |   yes  |   yes
6411    Arguments :
6412      <realm>   is the name of the HTTP Basic Authentication realm reported to
6413                the browser. The browser uses it to display it in the pop-up
6414                inviting the user to enter a valid username and password.
6415
6416    The realm is read as a single word, so any spaces in it should be escaped
6417    using a backslash ('\').
6418
6419    This statement is useful only in conjunction with "stats auth" since it is
6420    only related to authentication.
6421
6422    Though this statement alone is enough to enable statistics reporting, it is
6423    recommended to set all other settings in order to avoid relying on default
6424    unobvious parameters.
6425
6426    Example :
6427      # public access (limited to this backend only)
6428      backend public_www
6429          server srv1 192.168.0.1:80
6430          stats enable
6431          stats hide-version
6432          stats scope   .
6433          stats uri     /admin?stats
6434          stats realm   Haproxy\ Statistics
6435          stats auth    admin1:AdMiN123
```



```
6436          stats auth    admin2:AdMiN321
6437
6438      # internal monitoring access (unlimited)
6439      backend private_monitoring
6440          stats enable
6441          stats uri     /admin?stats
6442          stats refresh 5s
6443
6444    See also : "stats auth", "stats enable", "stats uri"
6445
6446
6447  stats refresh <delay>
6448    Enable statistics with automatic refresh
6449    May be used in sections :   defaults | frontend | listen | backend
6450                                   yes   |    yes   |   yes  |   yes
6451    Arguments :
6452      <delay>   is the suggested refresh delay, specified in seconds, which will
6453                be returned to the browser consulting the report page. While the
6454                browser is free to apply any delay, it will generally respect it
6455                and refresh the page this every seconds. The refresh interval may
6456                be specified in any other non-default time unit, by suffixing the
6457                unit after the value, as explained at the top of this document.
6458
6459    This statement is useful on monitoring displays with a permanent page
6460    reporting the load balancer's activity. When set, the HTML report page will
6461    include a link "refresh"/"stop refresh" so that the user can select whether
6462    he wants automatic refresh of the page or not.
6463
6464    Though this statement alone is enough to enable statistics reporting, it is
6465    recommended to set all other settings in order to avoid relying on default
6466    unobvious parameters.
6467
6468    Example :
6469      # public access (limited to this backend only)
6470      backend public_www
6471          server srv1 192.168.0.1:80
6472          stats enable
6473          stats hide-version
6474          stats scope   .
6475          stats uri     /admin?stats
6476          stats realm   Haproxy\ Statistics
6477          stats auth    admin1:AdMiN123
6478          stats auth    admin2:AdMiN321
6479
6480      # internal monitoring access (unlimited)
6481      backend private_monitoring
6482          stats enable
6483          stats uri     /admin?stats
6484          stats refresh 5s
6485
6486    See also : "stats auth", "stats enable", "stats realm", "stats uri"
6487
6488
6489  stats scope { <name> | "." }
6490    Enable statistics and limit access scope
6491    May be used in sections :   defaults | frontend | listen | backend
6492                                   yes   |    yes   |   yes  |   yes
6493    Arguments :
6494      <name>    is the name of a listen, frontend or backend section to be
6495                reported. The special name "." (a single dot) designates the
6496                section in which the statement appears.
6497
6498    When this statement is specified, only the sections enumerated with this
6499    statement will appear in the report. All other ones will be hidden. This
6500    statement may appear as many times as needed if multiple sections need to be
```

```
   reported. Please note that the name checking is performed as simple string
   comparisons, and that it is never checked that a give section name really
   exists.

   Though this statement alone is enough to enable statistics reporting, it is
   recommended to set all other settings in order to avoid relying on default
   unobvious parameters.

   Example :
     # public access (limited to this backend only)
     backend public_www
         server srv1 192.168.0.1:80
         stats enable
         stats hide-version
         stats scope   .
         stats uri     /admin?stats
         stats realm   Haproxy\ Statistics
         stats auth    admin1:AdMiN123
         stats auth    admin2:AdMiN321

     # internal monitoring access (unlimited)
     backend private_monitoring
         stats enable
         stats uri     /admin?stats
         stats refresh 5s

   See also : "stats auth", "stats enable", "stats realm", "stats uri"


stats show-desc [ <desc> ]
  Enable reporting of a description on the statistics page.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes

    <desc>    is an optional description to be reported. If unspecified, the
              description from global section is automatically used instead.

   This statement is useful for users that offer shared services to their
   customers, where node or description should be different for each customer.

   Though this statement alone is enough to enable statistics reporting, it is
   recommended to set all other settings in order to avoid relying on default
   unobvious parameters.  By default description is not shown.

   Example :
     # internal monitoring access (unlimited)
     backend private_monitoring
         stats enable
         stats show-desc Master node for Europe, Asia, Africa
         stats uri       /admin?stats
         stats refresh   5s

   See also: "show-node", "stats enable", "stats uri" and "description" in
             global section.


stats show-legends
  Enable reporting additional information on the statistics page
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments : none

  Enable reporting additional information on the statistics page :
    - cap: capabilities (proxy)
    - mode: one of tcp, http or health (proxy)
```



```
    - id: SNMP ID (proxy, socket, server)
    - IP (socket, server)
    - cookie (backend, server)

  Though this statement alone is enough to enable statistics reporting, it is
  recommended to set all other settings in order to avoid relying on default
  unobvious parameters.  Default behaviour is not to show this information.

  See also: "stats enable", "stats uri".


stats show-node [ <name> ]
  Enable reporting of a host name on the statistics page.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments:
    <name>    is an optional name to be reported. If unspecified, the
              node name from global section is automatically used instead.

  This statement is useful for users that offer shared services to their
  customers, where node or description might be different on a stats page
  provided for each customer.  Default behaviour is not to show host name.

  Though this statement alone is enough to enable statistics reporting, it is
  recommended to set all other settings in order to avoid relying on default
  unobvious parameters.

  Example:
    # internal monitoring access (unlimited)
    backend private_monitoring
        stats enable
        stats show-node Europe-1
        stats uri       /admin?stats
        stats refresh   5s

  See also: "show-desc", "stats enable", "stats uri", and "node" in global
            section.


stats uri <prefix>
  Enable statistics and define the URI prefix to access them
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   yes
  Arguments :
    <prefix>  is the prefix of any URI which will be redirected to stats. This
              prefix may contain a question mark ('?') to indicate part of a
              query string.

  The statistics URI is intercepted on the relayed traffic, so it appears as a
  page within the normal application. It is strongly advised to ensure that the
  selected URI will never appear in the application, otherwise it will never be
  possible to reach it in the application.

  The default URI compiled in haproxy is "/haproxy?stats", but this may be
  changed at build time, so it's better to always explicitly specify it here.
  It is generally a good idea to include a question mark in the URI so that
  intermediate proxies refrain from caching the results. Also, since any string
  beginning with the prefix will be accepted as a stats request, the question
  mark helps ensuring that no valid URI will begin with the same words.

  It is sometimes very convenient to use "/" as the URI prefix, and put that
  statement in a "listen" instance of its own. That makes it easy to dedicate
  an address or a port to statistics only.

  Though this statement alone is enough to enable statistics reporting, it is
```

```
   recommended to set all other settings in order to avoid relying on default
   unobvious parameters.

   Example :
     # public access (limited to this backend only)
     backend public_www
         server srv1 192.168.0.1:80
         stats enable
         stats hide-version
         stats scope   .
         stats uri     /admin?stats
         stats realm   Haproxy\ Statistics
         stats auth    admin1:AdMiN123
         stats auth    admin2:AdMiN321

     # internal monitoring access (unlimited)
     backend private_monitoring
         stats enable
         stats uri     /admin?stats
         stats refresh 5s

   See also : "stats auth", "stats enable", "stats realm"


stick match <pattern> [table <table>] [{if | unless} <cond>]
  Define a request pattern matching condition to stick a user to a server
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    no    |   yes  |   yes

  Arguments :
    <pattern>  is a sample expression rule as described in section 7.3. It
               describes what elements of the incoming request or connection
               will be analysed in the hope to find a matching entry in a
               stickiness table. This rule is mandatory.

    <table>    is an optional stickiness table name. If unspecified, the same
               backend's table is used. A stickiness table is declared using
               the "stick-table" statement.

    <cond>     is an optional matching condition. It makes it possible to match
               on a certain criterion only when other conditions are met (or
               not met). For instance, it could be used to match on a source IP
               address except when a request passes through a known proxy, in
               which case we'd match on a header containing that IP address.

  Some protocols or applications require complex stickiness rules and cannot
  always simply rely on cookies nor hashing. The "stick match" statement
  describes a rule to extract the stickiness criterion from an incoming request
  or connection. See section 7 for a complete list of possible patterns and
  transformation rules.

  The table has to be declared using the "stick-table" statement. It must be of
  a type compatible with the pattern. By default it is the one which is present
  in the same backend. It is possible to share a table with other backends by
  referencing it using the "table" keyword. If another table is referenced,
  the server's ID inside the backends are used. By default, all server IDs
  start at 1 in each backend, so the server ordering is enough. But in case of
  doubt, it is highly recommended to force server IDs using their "id" setting.

  It is possible to restrict the conditions where a "stick match" statement
  will apply, using "if" or "unless" followed by a condition. See section 7 for
  ACL based conditions.

  There is no limit on the number of "stick match" statements. The first that
  applies and matches will cause the request to be directed to the same server
```



```
  as was used for the request which created the entry. That way, multiple
  matches can be used as fallbacks.

  The stick rules are checked after the persistence cookies, so they will not
  affect stickiness if a cookie has already been used to select a server. That
  way, it becomes very easy to insert cookies and match on IP addresses in
  order to maintain stickiness between HTTP and HTTPS.

  Note : Consider not using this feature in multi-process mode (nbproc > 1)
         unless you know what you do : memory is not shared between the
         processes, which can result in random behaviours.

  Example :
    # forward SMTP users to the same server they just used for POP in the
    # last 30 minutes
    backend pop
        mode tcp
        balance roundrobin
        stick store-request src
        stick-table type ip size 200k expire 30m
        server s1 192.168.1.1:110
        server s2 192.168.1.1:110

    backend smtp
        mode tcp
        balance roundrobin
        stick match src table pop
        server s1 192.168.1.1:25
        server s2 192.168.1.1:25

  See also : "stick-table", "stick on", "nbproc", "bind-process" and section 7
             about ACLs and samples fetching.


stick on <pattern> [table <table>] [{if | unless} <condition>]
  Define a request pattern to associate a user to a server
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    no    |   yes  |   yes

  Note : This form is exactly equivalent to "stick match" followed by
         "stick store-request", all with the same arguments. Please refer
         to both keywords for details. It is only provided as a convenience
         for writing more maintainable configurations.

  Note : Consider not using this feature in multi-process mode (nbproc > 1)
         unless you know what you do : memory is not shared between the
         processes, which can result in random behaviours.

  Examples :
    # The following form ...
    stick on src table pop if !localhost

    # ...is strictly equivalent to this one :
    stick match src table pop if !localhost
    stick store-request src table pop if !localhost


    # Use cookie persistence for HTTP, and stick on source address for HTTPS as
    # well as HTTP without cookie. Share the same table between both accesses.
    backend http
        mode http
        balance roundrobin
        stick on src table https
        cookie SRV insert indirect nocache
        server s1 192.168.1.1:80 cookie s1
```

```
6761          server s2 192.168.1.1:80 cookie s2
6762
6763      backend https
6764          mode tcp
6765          balance roundrobin
6766          stick-table type ip size 200k expire 30m
6767          stick on src
6768          server s1 192.168.1.1:443
6769          server s2 192.168.1.1:443
6770
6771    See also : "stick match", "stick store-request", "nbproc" and "bind-process".
6772
6773
6774  stick store-request <pattern> [table <table>] [{if | unless} <condition>]
6775    Define a request pattern used to create an entry in a stickiness table
6776    May be used in sections :   defaults | frontend | listen | backend
6777                                   no    |    no    |   yes  |   yes
6778
6779    Arguments :
6780      <pattern>  is a sample expression rule as described in section 7.3. It
6781                 describes what elements of the incoming request or connection
6782                 will be analysed, extracted and stored in the table once a
6783                 server is selected.
6784
6785      <table>    is an optional stickiness table name. If unspecified, the same
6786                 backend's table is used. A stickiness table is declared using
6787                 the "stick-table" statement.
6788
6789      <cond>     is an optional storage condition. It makes it possible to store
6790                 certain criteria only when some conditions are met (or not met).
6791                 For instance, it could be used to store the source IP address
6792                 except when the request passes through a known proxy, in which
6793                 case we'd store a converted form of a header containing that IP
6794                 address.
6795
6796    Some protocols or applications require complex stickiness rules and cannot
6797    always simply rely on cookies nor hashing. The "stick store-request" statement
6798    describes a rule to decide what to extract from the request and when to do
6799    it, in order to store it into a stickiness table for further requests to
6800    match it using the "stick match" statement. Obviously the extracted part must
6801    make sense and have a chance to be matched in a further request. Storing a
6802    client's IP address for instance often makes sense. Storing an ID found in a
6803    URL parameter also makes sense. Storing a source port will almost never make
6804    any sense because it will be randomly matched. See section 7 for a complete
6805    list of possible patterns and transformation rules.
6806
6807    The table has to be declared using the "stick-table" statement. It must be of
6808    a type compatible with the pattern. By default it is the one which is present
6809    in the same backend. It is possible to share a table with other backends by
6810    referencing it using the "table" keyword. If another table is referenced,
6811    the server's ID inside the backends are used. By default, all server IDs
6812    start at 1 in each backend, so the server ordering is enough. But in case of
6813    doubt, it is highly recommended to force server IDs using their "id" setting.
6814
6815    It is possible to restrict the conditions where a "stick store-request"
6816    statement will apply, using "if" or "unless" followed by a condition. This
6817    condition will be evaluated while parsing the request, so any criteria can be
6818    used. See section 7 for ACL based conditions.
6819
6820    There is no limit on the number of "stick store-request" statements, but
6821    there is a limit of 8 simultaneous stores per request or response. This
6822    makes it possible to store up to 8 criteria, all extracted from either the
6823    request or the response, regardless of the number of rules. Only the 8 first
6824    ones which match will be kept. Using this, it is possible to feed multiple
6825    tables at once in the hope to increase the chance to recognize a user on
```



```
6826    another protocol or access method. Using multiple store-request rules with
6827    the same table is possible and may be used to find the best criterion to rely
6828    on, by arranging the rules by decreasing preference order. Only the first
6829    extracted criterion for a given table will be stored. All subsequent store-
6830    request rules referencing the same table will be skipped and their ACLs will
6831    not be evaluated.
6832
6833    The "store-request" rules are evaluated once the server connection has been
6834    established, so that the table will contain the real server that processed
6835    the request.
6836
6837    Note : Consider not using this feature in multi-process mode (nbproc > 1)
6838           unless you know what you do : memory is not shared between the
6839           processes, which can result in random behaviours.
6840
6841    Example :
6842      # forward SMTP users to the same server they just used for POP in the
6843      # last 30 minutes
6844      backend pop
6845          mode tcp
6846          balance roundrobin
6847          stick store-request src
6848          stick-table type ip size 200k expire 30m
6849          server s1 192.168.1.1:110
6850          server s2 192.168.1.1:110
6851
6852      backend smtp
6853          mode tcp
6854          balance roundrobin
6855          stick match src table pop
6856          server s1 192.168.1.1:25
6857          server s2 192.168.1.1:25
6858
6859    See also : "stick-table", "stick on", "nbproc", "bind-process" and section 7
6860               about ACLs and sample fetching.
6861
6862
6863  stick-table type {ip | integer | string [len <length>] | binary [len <length>]}
6864              size <size> [expire <expire>] [nopurge] [peers <peersect>]
6865              [store <data_type>]*
6866    Configure the stickiness table for the current section
6867    May be used in sections :   defaults | frontend | listen | backend
6868                                   no    |    yes   |   yes  |   yes
6869
6870    Arguments :
6871      ip         a table declared with "type ip" will only store IPv4 addresses.
6872                 This form is very compact (about 50 bytes per entry) and allows
6873                 very fast entry lookup and stores with almost no overhead. This
6874                 is mainly used to store client source IP addresses.
6875
6876      ipv6       a table declared with "type ipv6" will only store IPv6 addresses.
6877                 This form is very compact (about 60 bytes per entry) and allows
6878                 very fast entry lookup and stores with almost no overhead. This
6879                 is mainly used to store client source IP addresses.
6880
6881      integer    a table declared with "type integer" will store 32bit integers
6882                 which can represent a client identifier found in a request for
6883                 instance.
6884
6885      string     a table declared with "type string" will store substrings of up
6886                 to <len> characters. If the string provided by the pattern
6887                 extractor is larger than <len>, it will be truncated before
6888                 being stored. During matching, at most <len> characters will be
6889                 compared between the string in the table and the extracted
6890                 pattern. When not specified, the string is automatically limited
```

to 32 characters.

binary a table declared with "type binary" will store binary blocks of <len> bytes. If the block provided by the pattern extractor is larger than <len>, it will be truncated before being stored. If the block provided by the sample expression is shorter than <len>, it will be padded by 0. When not specified, the block is automatically limited to 32 bytes.

<length> is the maximum number of characters that will be stored in a "string" type table (See type "string" above). Or the number of bytes of the block in "binary" type table. Be careful when changing this parameter as memory usage will proportionally increase.

<size> is the maximum number of entries that can fit in the table. This value directly impacts memory usage. Count approximately 50 bytes per entry, plus the size of a string if any. The size supports suffixes "k", "m", "g" for 2^10, 2^20 and 2^30 factors.

[nopurge] indicates that we refuse to purge older entries when the table is full. When not specified and the table is full when haproxy wants to store an entry in it, it will flush a few of the oldest entries in order to release some space for the new ones. This is most often the desired behaviour. In some specific cases, it be desirable to refuse new entries instead of purging the older ones. That may be the case when the amount of data to store is far above the hardware limits and we prefer not to offer access to new clients than to reject the ones already connected. When using this parameter, be sure to properly set the "expire" parameter (see below).

<peersect> is the name of the peers section to use for replication. Entries which associate keys to server IDs are kept synchronized with the remote peers declared in this section. All entries are also automatically learned from the local peer (old process) during a soft restart.

NOTE : each peers section may be referenced only by tables belonging to the same unique process.

<expire> defines the maximum duration of an entry in the table since it was last created, refreshed or matched. The expiration delay is defined using the standard time format, similarly as the various timeouts. The maximum duration is slightly above 24 days. See section 2.2 for more information. If this delay is not specified, the session won't automatically expire, but older entries will be removed once full. Be sure not to use the "nopurge" parameter if not expiration delay is specified.

<data_type> is used to store additional information in the stick-table. This may be used by ACLs in order to control various criteria related to the activity of the client matching the stick-table. For each item specified here, the size of each entry will be inflated so that the additional data can fit. Several data types may be stored with an entry. Multiple data types may be specified after the "store" keyword, as a comma-separated list. Alternatively, it is possible to repeat the "store" keyword followed by one or several data types. Except for the "server_id" type which is automatically detected and enabled, all data types must be explicitly declared to be stored. If an ACL references a data type which is not stored, the ACL will simply not match. Some data types require an argument which must be passed just after the type between parenthesis. See below for the supported data types and their arguments.

The data types that can be stored with an entry are the following :

- server_id : this is an integer which holds the numeric ID of the server a request was assigned to. It is used by the "stick match", "stick store", and "stick on" rules. It is automatically enabled when referenced.

- gpc0 : first General Purpose Counter. It is a positive 32-bit integer integer which may be used for anything. Most of the time it will be used to put a special tag on some entries, for instance to note that a specific behaviour was detected and must be known for future matches.

- gpc0_rate(<period>) : increment rate of the first General Purpose Counter over a period. It is a positive 32-bit integer integer which may be used for anything. Just like <gpc0>, it counts events, but instead of keeping a cumulative count, it maintains the rate at which the counter is incremented. Most of the time it will be used to measure the frequency of occurrence of certain events (eg: requests to a specific URL).

- conn_cnt : Connection Count. It is a positive 32-bit integer which counts the absolute number of connections received from clients which matched this entry. It does not mean the connections were accepted, just that they were received.

- conn_cur : Current Connections. It is a positive 32-bit integer which stores the concurrent connection counts for the entry. It is incremented once an incoming connection matches the entry, and decremented once the connection leaves. That way it is possible to know at any time the exact number of concurrent connections for an entry.

- conn_rate(<period>) : frequency counter (takes 12 bytes). It takes an integer parameter <period> which indicates in milliseconds the length of the period over which the average is measured. It reports the average incoming connection rate over that period, in connections per period. The result is an integer which can be matched using ACLs.

- sess_cnt : Session Count. It is a positive 32-bit integer which counts the absolute number of sessions received from clients which matched this entry. A session is a connection that was accepted by the layer 4 rules.

- sess_rate(<period>) : frequency counter (takes 12 bytes). It takes an integer parameter <period> which indicates in milliseconds the length of the period over which the average is measured. It reports the average incoming session rate over that period, in sessions per period. The result is an integer which can be matched using ACLs.

- http_req_cnt : HTTP request Count. It is a positive 32-bit integer which counts the absolute number of HTTP requests received from clients which matched this entry. It does not matter whether they are valid requests or not. Note that this is different from sessions when keep-alive is used on the client side.

- http_req_rate(<period>) : frequency counter (takes 12 bytes). It takes an integer parameter <period> which indicates in milliseconds the length of the period over which the average is measured. It reports the average HTTP request rate over that period, in requests per period. The result is an integer which can be matched using ACLs. It does not matter whether they are valid requests or not. Note that this is different from sessions when keep-alive is used on the client side.

- http_err_cnt : HTTP Error Count. It is a positive 32-bit integer which counts the absolute number of HTTP requests errors induced by clients which matched this entry. Errors are counted on invalid and truncated requests, as well as on denied or tarpitted requests, and on failed authentications. If the server responds with 4xx, then the request is also counted as an error since it's an error triggered by the client

```
7021        (eg: vulnerability scan).
7022
7023      - http_err_rate(<period>) : frequency counter (takes 12 bytes). It takes an
7024        integer parameter <period> which indicates in milliseconds the length
7025        of the period over which the average is measured. It reports the average
7026        HTTP request error rate over that period, in requests per period (see
7027        http_err_cnt above for what is accounted as an error). The result is an
7028        integer which can be matched using ACLs.
7029
7030      - bytes_in_cnt : client to server byte count. It is a positive 64-bit
7031        integer which counts the cumulated amount of bytes received from clients
7032        which matched this entry. Headers are included in the count. This may be
7033        used to limit abuse of upload features on photo or video servers.
7034
7035      - bytes_in_rate(<period>) : frequency counter (takes 12 bytes). It takes an
7036        integer parameter <period> which indicates in milliseconds the length
7037        of the period over which the average is measured. It reports the average
7038        incoming bytes rate over that period, in bytes per period. It may be used
7039        to detect users which upload too much and too fast. Warning: with large
7040        uploads, it is possible that the amount of uploaded data will be counted
7041        once upon termination, thus causing spikes in the average transfer speed
7042        instead of having a smooth one. This may partially be smoothed with
7043        "option contstats" though this is not perfect yet. Use of byte_in_cnt is
7044        recommended for better fairness.
7045
7046      - bytes_out_cnt : server to client byte count. It is a positive 64-bit
7047        integer which counts the cumulated amount of bytes sent to clients which
7048        matched this entry. Headers are included in the count. This may be used
7049        to limit abuse of bots sucking the whole site.
7050
7051      - bytes_out_rate(<period>) : frequency counter (takes 12 bytes). It takes
7052        an integer parameter <period> which indicates in milliseconds the length
7053        of the period over which the average is measured. It reports the average
7054        outgoing bytes rate over that period, in bytes per period. It may be used
7055        to detect users which download too much and too fast. Warning: with large
7056        transfers, it is possible that the amount of transferred data will be
7057        counted once upon termination, thus causing spikes in the average
7058        transfer speed instead of having a smooth one. This may partially be
7059        smoothed with "option contstats" though this is not perfect yet. Use of
7060        byte_out_cnt is recommended for better fairness.
7061
7062    There is only one stick-table per proxy. At the moment of writing this doc,
7063    it does not seem useful to have multiple tables per proxy. If this happens
7064    to be required, simply create a dummy backend with a stick-table in it and
7065    reference it.
7066
7067    It is important to understand that stickiness based on learning information
7068    has some limitations, including the fact that all learned associations are
7069    lost upon restart. In general it can be good as a complement but not always
7070    as an exclusive stickiness.
7071
7072    Last, memory requirements may be important when storing many data types.
7073    Indeed, storing all indicators above at once in each entry requires 116 bytes
7074    per entry, or 116 MB for a 1-million entries table. This is definitely not
7075    something that can be ignored.
7076
7077    Example:
7078          # Keep track of counters of up to 1 million IP addresses over 5 minutes
7079          # and store a general purpose counter and the average connection rate
7080          # computed over a sliding window of 30 seconds.
7081          stick-table type ip size 1m expire 5m store gpc0,conn_rate(30s)
7082
7083    See also : "stick match", "stick on", "stick store-request", section 2.2
7084               about time format and section 7 about ACLs.
7085
```



```
7086
7087  stick store-response <pattern> [table <table>] [{if | unless} <condition>]
7088    Define a request pattern used to create an entry in a stickiness table
7089    May be used in sections :   defaults | frontend | listen | backend
7090                                   no    |    no    |   yes  |   yes
7091
7092    Arguments :
7093      <pattern>  is a sample expression rule as described in section 7.3. It
7094                 describes what elements of the response or connection will
7095                 be analysed, extracted and stored in the table once a
7096                 server is selected.
7097
7098      <table>    is an optional stickiness table name. If unspecified, the same
7099                 backend's table is used. A stickiness table is declared using
7100                 the "stick-table" statement.
7101
7102      <cond>     is an optional storage condition. It makes it possible to store
7103                 certain criteria only when some conditions are met (or not met).
7104                 For instance, it could be used to store the SSL session ID only
7105                 when the response is a SSL server hello.
7106
7107    Some protocols or applications require complex stickiness rules and cannot
7108    always simply rely on cookies nor hashing. The "stick store-response"
7109    statement  describes a rule to decide what to extract from the response and
7110    when to do it, in order to store it into a stickiness table for further
7111    requests to match it using the "stick match" statement. Obviously the
7112    extracted part must make sense and have a chance to be matched in a further
7113    request. Storing an ID found in a header of a response makes sense.
7114    See section 7 for a complete list of possible patterns and transformation
7115    rules.
7116
7117    The table has to be declared using the "stick-table" statement. It must be of
7118    a type compatible with the pattern. By default it is the one which is present
7119    in the same backend. It is possible to share a table with other backends by
7120    referencing it using the "table" keyword. If another table is referenced,
7121    the server's ID inside the backends are used. By default, all server IDs
7122    start at 1 in each backend, so the server ordering is enough. But in case of
7123    doubt, it is highly recommended to force server IDs using their "id" setting.
7124
7125    It is possible to restrict the conditions where a "stick store-response"
7126    statement will apply, using "if" or "unless" followed by a condition. This
7127    condition will be evaluated while parsing the response, so any criteria can
7128    be used. See section 7 for ACL based conditions.
7129
7130    There is no limit on the number of "stick store-response" statements, but
7131    there is a limit of 8 simultaneous stores per request or response. This
7132    makes it possible to store up to 8 criteria, all extracted from either the
7133    request or the response, regardless of the number of rules. Only the 8 first
7134    ones which match will be kept. Using this, it is possible to feed multiple
7135    tables at once in the hope to increase the chance to recognize a user on
7136    another protocol or access method. Using multiple store-response rules with
7137    the same table is possible and may be used to find the best criterion to rely
7138    on, by arranging the rules by decreasing preference order. Only the first
7139    extracted criterion for a given table will be stored. All subsequent store-
7140    response rules referencing the same table will be skipped and their ACLs will
7141    not be evaluated. However, even if a store-request rule references a table, a
7142    store-response rule may also use the same table. This means that each table
7143    may learn exactly one element from the request and one element from the
7144    response at once.
7145
7146    The table will contain the real server that processed the request.
7147
7148    Example :
7149      # Learn SSL session ID from both request and response and create affinity.
7150      backend https
```

```
7151          mode tcp
7152          balance roundrobin
7153          # maximum SSL session ID length is 32 bytes.
7154          stick-table type binary len 32 size 30k expire 30m
7155
7156          acl clienthello req_ssl_hello_type 1
7157          acl serverhello rep_ssl_hello_type 2
7158
7159          # use tcp content accepts to detects ssl client and server hello.
7160          tcp-request inspect-delay 5s
7161          tcp-request content accept if clienthello
7162
7163          # no timeout on response inspect delay by default.
7164          tcp-response content accept if serverhello
7165
7166          # SSL session ID (SSLID) may be present on a client or server hello.
7167          # Its length is coded on 1 byte at offset 43 and its value starts
7168          # at offset 44.
7169
7170          # Match and learn on request if client hello.
7171          stick on payload_lv(43,1) if clienthello
7172
7173          # Learn on response if server hello.
7174          stick store-response payload_lv(43,1) if serverhello
7175
7176          server s1 192.168.1.1:443
7177          server s2 192.168.1.1:443
7178
7179    See also : "stick-table", "stick on", and section 7 about ACLs and pattern
7180               extraction.
7181
7182
7183  tcp-check connect [params*]
7184    Opens a new connection
7185    May be used in sections:   defaults | frontend | listen | backend
7186                                  no     |    no    |   yes  |   yes
7187
7188    When an application lies on more than a single TCP port or when HAProxy
7189    load-balance many services in a single backend, it makes sense to probe all
7190    the services individually before considering a server as operational.
7191
7192    When there are no TCP port configured on the server line neither server port
7193    directive, then the 'tcp-check connect port <port>' must be the first step
7194    of the sequence.
7195
7196    In a tcp-check ruleset a 'connect' is required, it is also mandatory to start
7197    the ruleset with a 'connect' rule. Purpose is to ensure admin know what they
7198    do.
7199
7200    Parameters :
7201      They are optional and can be used to describe how HAProxy should open and
7202      use the TCP connection.
7203
7204      port      if not set, check port or server port is used.
7205                It tells HAProxy where to open the connection to.
7206                <port> must be a valid TCP port source integer, from 1 to 65535.
7207
7208      send-proxy   send a PROXY protocol string
7209
7210      ssl          opens a ciphered connection
7211
7212      Examples:
7213           # check HTTP and HTTPs services on a server.
7214           # first open port 80 thanks to server line port directive, then
7215           # tcp-check opens port 443, ciphered and run a request on it:
```



```
7216          option tcp-check
7217          tcp-check connect
7218          tcp-check send GET\ /\ HTTP/1.0\r\n
7219          tcp-check send Host:\ haproxy.1wt.eu\r\n
7220          tcp-check send \r\n
7221          tcp-check expect rstring (2..|3..)
7222          tcp-check connect port 443 ssl
7223          tcp-check send GET\ /\ HTTP/1.0\r\n
7224          tcp-check send Host:\ haproxy.1wt.eu\r\n
7225          tcp-check send \r\n
7226          tcp-check expect rstring (2..|3..)
7227          server www 10.0.0.1 check port 80
7228
7229          # check both POP and IMAP from a single server:
7230          option tcp-check
7231          tcp-check connect port 110
7232          tcp-check expect string +OK\ POP3\ ready
7233          tcp-check connect port 143
7234          tcp-check expect string *\ OK\ IMAP4\ ready
7235          server mail 10.0.0.1 check
7236
7237    See also : "option tcp-check", "tcp-check send", "tcp-check expect"
7238
7239
7240  tcp-check expect [!] <match> <pattern>
7241    Specify data to be collected and analysed during a generic health check
7242    May be used in sections:   defaults | frontend | listen | backend
7243                                  no     |    no    |   yes  |   yes
7244
7245    Arguments :
7246      <match>   is a keyword indicating how to look for a specific pattern in the
7247                response. The keyword may be one of "string", "rstring" or
7248                binary.
7249                The keyword may be preceded by an exclamation mark ("!") to negate
7250                the match. Spaces are allowed between the exclamation mark and the
7251                keyword. See below for more details on the supported keywords.
7252
7253      <pattern> is the pattern to look for. It may be a string or a regular
7254                expression. If the pattern contains spaces, they must be escaped
7255                with the usual backslash ('\').
7256                If the match is set to binary, then the pattern must be passed as
7257                a serie of hexadecimal digits in an even number. Each sequence of
7258                two digits will represent a byte. The hexadecimal digits may be
7259                used upper or lower case.
7260
7261
7262    The available matches are intentionally similar to their http-check cousins :
7263
7264      string <string> : test the exact string matches in the response buffer.
7265                        A health check response will be considered valid if the
7266                        response's buffer contains this exact string. If the
7267                        "string" keyword is prefixed with "!", then the response
7268                        will be considered invalid if the body contains this
7269                        string. This can be used to look for a mandatory pattern
7270                        in a protocol response, or to detect a failure when a
7271                        specific error appears in a protocol banner.
7272
7273      rstring <regex> : test a regular expression on the response buffer.
7274                        A health check response will be considered valid if the
7275                        response's buffer matches this expression. If the
7276                        "rstring" keyword is prefixed with "!", then the response
7277                        will be considered invalid if the body matches the
7278                        expression.
7279
7280      binary <hexstring> : test the exact string in its hexadecimal form matches
```

```
                           in the response buffer. A health check response will
                           be considered valid if the response's buffer contains
                           this exact hexadecimal string.
                           Purpose is to match data on binary protocols.

   It is important to note that the responses will be limited to a certain size
   defined by the global "tune.chksize" option, which defaults to 16384 bytes.
   Thus, too large responses may not contain the mandatory pattern when using
   "string", "rstring" or binary. If a large response is absolutely required, it
   is possible to change the default max size by setting the global variable.
   However, it is worth keeping in mind that parsing very large responses can
   waste some CPU cycles, especially when regular expressions are used, and that
   it is always better to focus the checks on smaller resources. Also, in its
   current state, the check will not find any string nor regex past a null
   character in the response. Similarly it is not possible to request matching
   the null character.

   Examples :
          # perform a POP check
          option tcp-check
          tcp-check expect string +OK\ POP3\ ready

          # perform an IMAP check
          option tcp-check
          tcp-check expect string *\ OK\ IMAP4\ ready

          # look for the redis master server
          option tcp-check
          tcp-check send PING\r\n
          tcp-check expect +PONG
          tcp-check send info\ replication\r\n
          tcp-check expect string role:master
          tcp-check send QUIT\r\n
          tcp-check expect string +OK


   See also : "option tcp-check", "tcp-check connect", "tcp-check send",
              "tcp-check send-binary", "http-check expect", tune.chksize


tcp-check send <data>
  Specify a string to be sent as a question during a generic health check
  May be used in sections:   defaults | frontend | listen | backend
                                no     |    no    |   yes  |   yes

    <data> : the data to be sent as a question during a generic health check
             session. For now, <data> must be a string.

   Examples :
          # look for the redis master server
          option tcp-check
          tcp-check send info\ replication\r\n
          tcp-check expect string role:master

   See also : "option tcp-check", "tcp-check connect", "tcp-check expect",
              "tcp-check send-binary", tune.chksize


tcp-check send-binary <hexastring>
  Specify an hexa digits string to be sent as a binary question during a raw
  tcp health check
  May be used in sections:   defaults | frontend | listen | backend
                                no     |    no    |   yes  |   yes

    <data> : the data to be sent as a question during a generic health check
```



```
             session. For now, <data> must be a string.
    <hexastring> : test the exact string in its hexadecimal form matches in the
                   response buffer. A health check response will be considered
                   valid if the response's buffer contains this exact
                   hexadecimal string.
                   Purpose is to send binary data to ask on binary protocols.

   Examples :
          # redis check in binary
          option tcp-check
          tcp-check send-binary 50494e470d0a # PING\r\n
          tcp-check expect binary 2b504F4e47 # +PONG


   See also : "option tcp-check", "tcp-check connect", "tcp-check expect",
              "tcp-check send", tune.chksize


tcp-request connection <action> [{if | unless} <condition>]
  Perform an action on an incoming connection depending on a layer 4 condition
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    yes   |   yes  |   no
  Arguments :
    <action>    defines the action to perform if the condition applies. Valid
                actions include : "accept", "reject", "track-sc0", "track-sc1",
                "track-sc2", and "expect-proxy". See below for more details.

    <condition> is a standard layer4-only ACL-based condition (see section 7).

  Immediately after acceptance of a new incoming connection, it is possible to
  evaluate some conditions to decide whether this connection must be accepted
  or dropped or have its counters tracked. Those conditions cannot make use of
  any data contents because the connection has not been read from yet, and the
  buffers are not yet allocated. This is used to selectively and very quickly
  accept or drop connections from various sources with a very low overhead. If
  some contents need to be inspected in order to take the decision, the
  "tcp-request content" statements must be used instead.

  The "tcp-request connection" rules are evaluated in their exact declaration
  order. If no rule matches or if there is no rule, the default action is to
  accept the incoming connection. There is no specific limit to the number of
  rules which may be inserted.

  Five types of actions are supported :
    - accept :
        accepts the connection if the condition is true (when used with "if")
        or false (when used with "unless"). The first such rule executed ends
        the rules evaluation.

    - reject :
        rejects the connection if the condition is true (when used with "if")
        or false (when used with "unless"). The first such rule executed ends
        the rules evaluation. Rejected connections do not even become a
        session, which is why they are accounted separately for in the stats,
        as "denied connections". They are not considered for the session
        rate-limit and are not logged either. The reason is that these rules
        should only be used to filter extremely high connection rates such as
        the ones encountered during a massive DDoS attack. Under these extreme
        conditions, the simple action of logging each event would make the
        system collapse and would considerably lower the filtering capacity. If
        logging is absolutely desired, then "tcp-request content" rules should
        be used instead.

    - expect-proxy layer4 :
        configures the client-facing connection to receive a PROXY protocol
```

```
          header before any byte is read from the socket. This is equivalent to
          having the "accept-proxy" keyword on the "bind" line, except that using
          the TCP rule allows the PROXY protocol to be accepted only for certain
          IP address ranges using an ACL. This is convenient when multiple layers
          of load balancers are passed through by traffic coming from public
          hosts.

      - capture <sample> len <length> :
          This only applies to "tcp-request content" rules. It captures sample
          expression <sample> from the request buffer, and converts it to a
          string of at most <len> characters. The resulting string is stored into
          the next request "capture" slot, so it will possibly appear next to
          some captured HTTP headers. It will then automatically appear in the
          logs, and it will be possible to extract it using sample fetch rules to
          feed it into headers or anything. The length should be limited given
          that this size will be allocated for each capture during the whole
          session life. Since it applies to Please check section 7.3 (Fetching
          samples) and "capture request header" for more information.

      - { track-sc0 | track-sc1 | track-sc2 } <key> [table <table>] :
          enables tracking of sticky counters from current connection. These
          rules do not stop evaluation and do not change default action. Two sets
          of counters may be simultaneously tracked by the same connection. The
          first "track-sc0" rule executed enables tracking of the counters of the
          specified table as the first set. The first "track-sc1" rule executed
          enables tracking of the counters of the specified table as the second
          set. The first "track-sc2" rule executed enables tracking of the
          counters of the specified table as the third set. It is a recommended
          practice to use the first set of counters for the per-frontend counters
          and the second set for the per-backend ones. But this is just a
          guideline, all may be used everywhere.

          These actions take one or two arguments :
            <key>   is mandatory, and is a sample expression rule as described
                    in section 7.3. It describes what elements of the incoming
                    request or connection will be analysed, extracted, combined,
                    and used to select which table entry to update the counters.
                    Note that "tcp-request connection" cannot use content-based
                    fetches.

           <table>  is an optional table to be used instead of the default one,
                    which is the stick-table declared in the current proxy. All
                    the counters for the matches and updates for the key will
                    then be performed in that table until the session ends.

          Once a "track-sc*" rule is executed, the key is looked up in the table
          and if it is not found, an entry is allocated for it. Then a pointer to
          that entry is kept during all the session's life, and this entry's
          counters are updated as often as possible, every time the session's
          counters are updated, and also systematically when the session ends.
          Counters are only updated for events that happen after the tracking has
          been started. For example, connection counters will not be updated when
          tracking layer 7 information, since the connection event happens before
          layer7 information is extracted.

          If the entry tracks concurrent connection counters, one connection is
          counted for as long as the entry is tracked, and the entry will not
          expire during that time. Tracking counters also provides a performance
          advantage over just checking the keys, because only one table lookup is
          performed for all ACL checks that make use of it.

    Note that the "if/unless" condition is optional. If no condition is set on
    the action, it is simply performed unconditionally. That can be useful for
    "track-sc*" actions as well as for changing the default action to a reject.

```



```
    Example: accept all connections from white-listed hosts, reject too fast
             connection without counting them, and track accepted connections.
             This results in connection rate being capped from abusive sources.

          tcp-request connection accept if { src -f /etc/haproxy/whitelist.lst }
          tcp-request connection reject if { src_conn_rate gt 10 }
          tcp-request connection track-sc0 src

    Example: accept all connections from white-listed hosts, count all other
             connections and reject too fast ones. This results in abusive ones
             being blocked as long as they don't slow down.

          tcp-request connection accept if { src -f /etc/haproxy/whitelist.lst }
          tcp-request connection track-sc0 src
          tcp-request connection reject if { sc0_conn_rate gt 10 }

    Example: enable the PROXY protocol for traffic coming from all known proxies.

          tcp-request connection expect-proxy layer4 if { src -f proxies.lst }

    See section 7 about ACL usage.

    See also : "tcp-request content", "stick-table"


  tcp-request content <action> [{if | unless} <condition>]
    Perform an action on a new session depending on a layer 4-7 condition
    May be used in sections :   defaults | frontend | listen | backend
                                   no    |    yes   |   yes  |   yes
    Arguments :
      <action>    defines the action to perform if the condition applies. Valid
                  actions include : "accept", "reject", "track-sc0", "track-sc1",
                  "track-sc2" and "capture". See "tcp-request connection" above
                  for their signification.

      <condition> is a standard layer 4-7 ACL-based condition (see section 7).

    A request's contents can be analysed at an early stage of request processing
    called "TCP content inspection". During this stage, ACL-based rules are
    evaluated every time the request contents are updated, until either an
    "accept" or a "reject" rule matches, or the TCP request inspection delay
    expires with no matching rule.

    The first difference between these rules and "tcp-request connection" rules
    is that "tcp-request content" rules can make use of contents to take a
    decision. Most often, these decisions will consider a protocol recognition or
    validity. The second difference is that content-based rules can be used in
    both frontends and backends. In case of HTTP keep-alive with the client, all
    tcp-request content rules are evaluated again, so haproxy keeps a record of
    what sticky counters were assigned by a "tcp-request connection" versus a
    "tcp-request content" rule, and flushes all the content-related ones after
    processing an HTTP request, so that they may be evaluated again by the rules
    being evaluated again for the next request. This is of particular importance
    when the rule tracks some L7 information or when it is conditioned by an
    L7-based ACL, since tracking may change between requests.

    Content-based rules are evaluated in their exact declaration order. If no
    rule matches or if there is no rule, the default action is to accept the
    contents. There is no specific limit to the number of rules which may be
    inserted.

    Four types of actions are supported :
      - accept : the request is accepted
      - reject : the request is rejected and the connection is closed
      - capture : the specified sample expression is captured
```

```
7541      - { track-sc0 | track-sc1 | track-sc2 } <key> [table <table>]
7542
7543    They have the same meaning as their counter-parts in "tcp-request connection"
7544    so please refer to that section for a complete description.
7545
7546    While there is nothing mandatory about it, it is recommended to use the
7547    track-sc0 in "tcp-request connection" rules, track-sc1 for "tcp-request
7548    content" rules in the frontend, and track-sc2 for "tcp-request content"
7549    rules in the backend, because that makes the configuration more readable
7550    and easier to troubleshoot, but this is just a guideline and all counters
7551    may be used everywhere.
7552
7553    Note that the "if/unless" condition is optional. If no condition is set on
7554    the action, it is simply performed unconditionally. That can be useful for
7555    "track-sc*" actions as well as for changing the default action to a reject.
7556
7557    It is perfectly possible to match layer 7 contents with "tcp-request content"
7558    rules, since HTTP-specific ACL matches are able to preliminarily parse the
7559    contents of a buffer before extracting the required data. If the buffered
7560    contents do not parse as a valid HTTP message, then the ACL does not match.
7561    The parser which is involved there is exactly the same as for all other HTTP
7562    processing, so there is no risk of parsing something differently. In an HTTP
7563    backend connected to from an HTTP frontend, it is guaranteed that HTTP
7564    contents will always be immediately present when the rule is evaluated first.
7565
7566    Tracking layer7 information is also possible provided that the information
7567    are present when the rule is processed. The rule processing engine is able to
7568    wait until the inspect delay expires when the data to be tracked is not yet
7569    available.
7570
7571    Example:
7572          # Accept HTTP requests containing a Host header saying "example.com"
7573          # and reject everything else.
7574          acl is_host_com hdr(Host) -i example.com
7575          tcp-request inspect-delay 30s
7576          tcp-request content accept if is_host_com
7577          tcp-request content reject
7578
7579    Example:
7580          # reject SMTP connection if client speaks first
7581          tcp-request inspect-delay 30s
7582          acl content_present req_len gt 0
7583          tcp-request content reject if content_present
7584
7585          # Forward HTTPS connection only if client speaks
7586          tcp-request inspect-delay 30s
7587          acl content_present req_len gt 0
7588          tcp-request content accept if content_present
7589          tcp-request content reject
7590
7591    Example:
7592          # Track the last IP from X-Forwarded-For
7593          tcp-request inspect-delay 10s
7594          tcp-request content track-sc0 hdr(x-forwarded-for,-1)
7595
7596    Example:
7597          # track request counts per "base" (concatenation of Host+URL)
7598          tcp-request inspect-delay 10s
7599          tcp-request content track-sc0 base table req-rate
7600
7601    Example: track per-frontend and per-backend counters, block abusers at the
7602             frontend when the backend detects abuse.
7603
7604          frontend http
7605              # Use General Purpose Couter 0 in SC0 as a global abuse counter
```



```
7606              # protecting all our sites
7607              stick-table type ip size 1m expire 5m store gpc0
7608              tcp-request connection track-sc0 src
7609              tcp-request connection reject if { sc0_get_gpc0 gt 0 }
7610              ...
7611              use_backend http_dynamic if { path_end .php }
7612
7613          backend http_dynamic
7614              # if a source makes too fast requests to this dynamic site (tracked
7615              # by SC1), block it globally in the frontend.
7616              stick-table type ip size 1m expire 5m store http_req_rate(10s)
7617              acl click_too_fast sc1_http_req_rate gt 10
7618              acl mark_as_abuser sc0_inc_gpc0 gt 0
7619              tcp-request content track-sc1 src
7620              tcp-request content reject if click_too_fast mark_as_abuser
7621
7622    See section 7 about ACL usage.
7623
7624    See also : "tcp-request connection", "tcp-request inspect-delay"
7625
7626
7627  tcp-request inspect-delay <timeout>
7628    Set the maximum allowed time to wait for data during content inspection
7629    May be used in sections :   defaults | frontend | listen | backend
7630                                   no    |    yes   |   yes  |   yes
7631    Arguments :
7632      <timeout> is the timeout value specified in milliseconds by default, but
7633                can be in any other unit if the number is suffixed by the unit,
7634                as explained at the top of this document.
7635
7636    People using haproxy primarily as a TCP relay are often worried about the
7637    risk of passing any type of protocol to a server without any analysis. In
7638    order to be able to analyze the request contents, we must first withhold
7639    the data then analyze them. This statement simply enables withholding of
7640    data for at most the specified amount of time.
7641
7642    TCP content inspection applies very early when a connection reaches a
7643    frontend, then very early when the connection is forwarded to a backend. This
7644    means that a connection may experience a first delay in the frontend and a
7645    second delay in the backend if both have tcp-request rules.
7646
7647    Note that when performing content inspection, haproxy will evaluate the whole
7648    rules for every new chunk which gets in, taking into account the fact that
7649    those data are partial. If no rule matches before the aforementioned delay,
7650    a last check is performed upon expiration, this time considering that the
7651    contents are definitive. If no delay is set, haproxy will not wait at all
7652    and will immediately apply a verdict based on the available information.
7653    Obviously this is unlikely to be very useful and might even be racy, so such
7654    setups are not recommended.
7655
7656    As soon as a rule matches, the request is released and continues as usual. If
7657    the timeout is reached and no rule matches, the default policy will be to let
7658    it pass through unaffected.
7659
7660    For most protocols, it is enough to set it to a few seconds, as most clients
7661    send the full request immediately upon connection. Add 3 or more seconds to
7662    cover TCP retransmits but that's all. For some protocols, it may make sense
7663    to use large values, for instance to ensure that the client never talks
7664    before the server (eg: SMTP), or to wait for a client to talk before passing
7665    data to the server (eg: SSL). Note that the client timeout must cover at
7666    least the inspection delay, otherwise it will expire first. If the client
7667    closes the connection or if the buffer is full, the delay immediately expires
7668    since the contents will not be able to change anymore.
7669
7670    See also : "tcp-request content accept", "tcp-request content reject",
```

```
              "timeout client".


tcp-response content <action> [{if | unless} <condition>]
  Perform an action on a session response depending on a layer 4-7 condition
  May be used in sections :   defaults | frontend | listen | backend
                                 no    |    no    |   yes  |   yes
  Arguments :
    <action>    defines the action to perform if the condition applies. Valid
                actions include : "accept", "close", "reject".

    <condition> is a standard layer 4-7 ACL-based condition (see section 7).

  Response contents can be analysed at an early stage of response processing
  called "TCP content inspection". During this stage, ACL-based rules are
  evaluated every time the response contents are updated, until either an
  "accept", "close" or a "reject" rule matches, or a TCP response inspection
  delay is set and expires with no matching rule.

  Most often, these decisions will consider a protocol recognition or validity.

  Content-based rules are evaluated in their exact declaration order. If no
  rule matches or if there is no rule, the default action is to accept the
  contents. There is no specific limit to the number of rules which may be
  inserted.

  Two types of actions are supported :
    - accept :
        accepts the response if the condition is true (when used with "if")
        or false (when used with "unless"). The first such rule executed ends
        the rules evaluation.

    - close :
        immediately closes the connection with the server if the condition is
        true (when used with "if"), or false (when used with "unless"). The
        first such rule executed ends the rules evaluation. The main purpose of
        this action is to force a connection to be finished between a client
        and a server after an exchange when the application protocol expects
        some long time outs to elapse first. The goal is to eliminate idle
        connections which take significant resources on servers with certain
        protocols.

    - reject :
        rejects the response if the condition is true (when used with "if")
        or false (when used with "unless"). The first such rule executed ends
        the rules evaluation. Rejected session are immediately closed.

  Note that the "if/unless" condition is optional. If no condition is set on
  the action, it is simply performed unconditionally. That can be useful for
  for changing the default action to a reject.

  It is perfectly possible to match layer 7 contents with "tcp-response
  content" rules, but then it is important to ensure that a full response has
  been buffered, otherwise no contents will match. In order to achieve this,
  the best solution involves detecting the HTTP protocol during the inspection
  period.

  See section 7 about ACL usage.

  See also : "tcp-request content", "tcp-response inspect-delay"


tcp-response inspect-delay <timeout>
  Set the maximum allowed time to wait for a response during content inspection
  May be used in sections :   defaults | frontend | listen | backend
```



```
                                 no    |    no    |   yes  |   yes
  Arguments :
    <timeout> is the timeout value specified in milliseconds by default, but
              can be in any other unit if the number is suffixed by the unit,
              as explained at the top of this document.

  See also : "tcp-response content", "tcp-request inspect-delay".


timeout check <timeout>
  Set additional check timeout, but only after a connection has been already
  established.

  May be used in sections:    defaults | frontend | listen | backend
                                 yes   |    no    |   yes  |   yes
  Arguments:
    <timeout> is the timeout value specified in milliseconds by default, but
              can be in any other unit if the number is suffixed by the unit,
              as explained at the top of this document.

  If set, haproxy uses min("timeout connect", "inter") as a connect timeout
  for check and "timeout check" as an additional read timeout. The "min" is
  used so that people running with *very* long "timeout connect" (eg. those
  who needed this due to the queue or tarpit) do not slow down their checks.
  (Please also note that there is no valid reason to have such long connect
  timeouts, because "timeout queue" and "timeout tarpit" can always be used to
  avoid that).

  If "timeout check" is not set haproxy uses "inter" for complete check
  timeout (connect + read) exactly like all <1.3.15 version.

  In most cases check request is much simpler and faster to handle than normal
  requests and people may want to kick out laggy servers so this timeout should
  be smaller than "timeout server".

  This parameter is specific to backends, but can be specified once for all in
  "defaults" sections. This is in fact one of the easiest solutions not to
  forget about it.

  See also: "timeout connect", "timeout queue", "timeout server",
            "timeout tarpit".


timeout client <timeout>
timeout clitimeout <timeout> (deprecated)
  Set the maximum inactivity time on the client side.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes   |    yes   |   yes  |   no
  Arguments :
    <timeout> is the timeout value specified in milliseconds by default, but
              can be in any other unit if the number is suffixed by the unit,
              as explained at the top of this document.

  The inactivity timeout applies when the client is expected to acknowledge or
  send data. In HTTP mode, this timeout is particularly important to consider
  during the first phase, when the client sends the request, and during the
  response while it is reading data sent by the server. The value is specified
  in milliseconds by default, but can be in any other unit if the number is
  suffixed by the unit, as specified at the top of this document. In TCP mode
  (and to a lesser extent, in HTTP mode), it is highly recommended that the
  client timeout remains equal to the server timeout in order to avoid complex
  situations to debug. It is a good practice to cover one or several TCP packet
  losses by specifying timeouts that are slightly above multiples of 3 seconds
  (eg: 4 or 5 seconds). If some long-lived sessions are mixed with short-lived
  sessions (eg: WebSocket and HTTP), it's worth considering "timeout tunnel",
```

which overrides "timeout client" and "timeout server" for tunnels, as well as
"timeout client-fin" for half-closed connections.

This parameter is specific to frontends, but can be specified once for all in
"defaults" sections. This is in fact one of the easiest solutions not to
forget about it. An unspecified timeout results in an infinite timeout, which
is not recommended. Such a usage is accepted and works but reports a warning
during startup because it may results in accumulation of expired sessions in
the system if the system's timeouts are not configured either.

This parameter replaces the old, deprecated "clitimeout". It is recommended
to use it to write new configurations. The form "timeout clitimeout" is
provided only by backwards compatibility but its use is strongly discouraged.

See also : "clitimeout", "timeout server", "timeout tunnel".

timeout client-fin <timeout>
  Set the inactivity timeout on the client side for half-closed connections.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes | yes | yes | no
  Arguments :
    <timeout> is the timeout value specified in milliseconds by default, but
              can be in any other unit if the number is suffixed by the unit,
              as explained at the top of this document.

  The inactivity timeout applies when the client is expected to acknowledge or
  send data while one direction is already shut down. This timeout is different
  from "timeout client" in that it only applies to connections which are closed
  in one direction. This is particularly useful to avoid keeping connections in
  FIN_WAIT state for too long when clients do not disconnect cleanly. This
  problem is particularly common long connections such as RDP or WebSocket.
  Note that this timeout can override "timeout tunnel" when a connection shuts
  down in one direction.

  This parameter is specific to frontends, but can be specified once for all in
  "defaults" sections. By default it is not set, so half-closed connections
  will use the other timeouts (timeout.client or timeout.tunnel).

  See also : "timeout client", "timeout server-fin", and "timeout tunnel".

timeout connect <timeout>
timeout contimeout <timeout> (deprecated)
  Set the maximum time to wait for a connection attempt to a server to succeed.
  May be used in sections :   defaults | frontend | listen | backend
                                 yes | no | yes | yes
  Arguments :
    <timeout> is the timeout value specified in milliseconds by default, but
              can be in any other unit if the number is suffixed by the unit,
              as explained at the top of this document.

  If the server is located on the same LAN as haproxy, the connection should be
  immediate (less than a few milliseconds). Anyway, it is a good practice to
  cover one or several TCP packet losses by specifying timeouts that are
  slightly above multiples of 3 seconds (eg: 4 or 5 seconds). By default, the
  connect timeout also presets both queue and tarpit timeouts to the same value
  if these have not been specified.

  This parameter is specific to backends, but can be specified once for all in
  "defaults" sections. This is in fact one of the easiest solutions not to
  forget about it. An unspecified timeout results in an infinite timeout, which
  is not recommended. Such a usage is accepted and works but reports a warning
  during startup because it may results in accumulation of failed sessions in
  the system if the system's timeouts are not configured either.

  This parameter replaces the old, deprecated "contimeout". It is recommended
  to use it to write new configurations. The form "timeout contimeout" is
  provided only by backwards compatibility but its use is strongly discouraged.

  See also: "timeout check", "timeout queue", "timeout server", "contimeout",
            "timeout tarpit".

timeout http-keep-alive <timeout>
  Set the maximum allowed time to wait for a new HTTP request to appear
  May be used in sections :   defaults | frontend | listen | backend
                                 yes | yes | yes | yes
  Arguments :
    <timeout> is the timeout value specified in milliseconds by default, but
              can be in any other unit if the number is suffixed by the unit,
              as explained at the top of this document.

  By default, the time to wait for a new request in case of keep-alive is set
  by "timeout http-request". However this is not always convenient because some
  people want very short keep-alive timeouts in order to release connections
  faster, and others prefer to have larger ones but still have short timeouts
  once the request has started to present itself.

  The "http-keep-alive" timeout covers these needs. It will define how long to
  wait for a new HTTP request to start coming after a response was sent. Once
  the first byte of request has been seen, the "http-request" timeout is used
  to wait for the complete request to come. Note that empty lines prior to a
  new request do not refresh the timeout and are not counted as a new request.

  There is also another difference between the two timeouts : when a connection
  expires during timeout http-keep-alive, no error is returned, the connection
  just closes. If the connection expires in "http-request" while waiting for a
  connection to complete, a HTTP 408 error is returned.

  In general it is optimal to set this value to a few tens to hundreds of
  milliseconds, to allow users to fetch all objects of a page at once but
  without waiting for further clicks. Also, if set to a very small value (eg:
  1 millisecond) it will probably only accept pipelined requests but not the
  non-pipelined ones. It may be a nice trade-off for very large sites running
  with tens to hundreds of thousands of clients.

  If this parameter is not set, the "http-request" timeout applies, and if both
  are not set, "timeout client" still applies at the lower level. It should be
  set in the frontend to take effect, unless the frontend is in TCP mode, in
  which case the HTTP backend's timeout will be used.

  See also : "timeout http-request", "timeout client".

timeout http-request <timeout>
  Set the maximum allowed time to wait for a complete HTTP request
  May be used in sections :   defaults | frontend | listen | backend
                                 yes | yes | yes | yes
  Arguments :
    <timeout> is the timeout value specified in milliseconds by default, but
              can be in any other unit if the number is suffixed by the unit,
              as explained at the top of this document.

  In order to offer DoS protection, it may be required to lower the maximum
  accepted time to receive a complete HTTP request without affecting the client
  timeout. This helps protecting against established connections on which
  nothing is sent. The client timeout cannot offer a good protection against
  this abuse because it is an inactivity timeout, which means that if the
  attacker sends one character every now and then, the timeout will not

trigger. With the HTTP request timeout, no matter what speed the client types, the request will be aborted if it does not complete in time. When the timeout expires, an HTTP 408 response is sent to the client to inform it about the problem, and the connection is closed. The logs will report termination codes "cR". Some recent browsers are having problems with this standard, well-documented behaviour, so it might be needed to hide the 408 code using "option http-ignore-probes" or "errorfile 408 /dev/null". See more details in the explanations of the "cR" termination code in section 8.5.

Note that this timeout only applies to the header part of the request, and not to any data. As soon as the empty line is received, this timeout is not used anymore. It is used again on keep-alive connections to wait for a second request if "timeout http-keep-alive" is not set.

Generally it is enough to set it to a few seconds, as most clients send the full request immediately upon connection. Add 3 or more seconds to cover TCP retransmits but that's all. Setting it to very low values (eg: 50 ms) will generally work on local networks as long as there are no packet losses. This will prevent people from sending bare HTTP requests using telnet.

If this parameter is not set, the client timeout still applies between each chunk of the incoming request. It should be set in the frontend to take effect, unless the frontend is in TCP mode, in which case the HTTP backend's timeout will be used.

See also : "errorfile", "http-ignore-probes", "timeout http-keep-alive", and "timeout client".

timeout queue <timeout>
Set the maximum time to wait in the queue for a connection slot to be free
May be used in sections : defaults | frontend | listen | backend
yes | no | yes | yes
Arguments :
<timeout> is the timeout value specified in milliseconds by default, but can be in any other unit if the number is suffixed by the unit, as explained at the top of this document.

When a server's maxconn is reached, connections are left pending in a queue which may be server-specific or global to the backend. In order not to wait indefinitely, a timeout is applied to requests pending in the queue. If the timeout is reached, it is considered that the request will almost never be served, so it is dropped and a 503 error is returned to the client.

The "timeout queue" statement allows to fix the maximum time for a request to be left pending in a queue. If unspecified, the same value as the backend's connection timeout ("timeout connect") is used, for backwards compatibility with older versions with no "timeout queue" parameter.

See also : "timeout connect", "contimeout".

timeout server <timeout>
timeout srvtimeout <timeout> (deprecated)
Set the maximum inactivity time on the server side.
May be used in sections : defaults | frontend | listen | backend
yes | no | yes | yes
Arguments :
<timeout> is the timeout value specified in milliseconds by default, but can be in any other unit if the number is suffixed by the unit, as explained at the top of this document.

The inactivity timeout applies when the server is expected to acknowledge or send data. In HTTP mode, this timeout is particularly important to consider during the first phase of the server's response, when it has to send the

headers, as it directly represents the server's processing time for the request. To find out what value to put there, it's often good to start with what would be considered as unacceptable response times, then check the logs to observe the response time distribution, and adjust the value accordingly.

The value is specified in milliseconds by default, but can be in any other unit if the number is suffixed by the unit, as specified at the top of this document. In TCP mode (and to a lesser extent, in HTTP mode), it is highly recommended that the client timeout remains equal to the server timeout in order to avoid complex situations to debug. Whatever the expected server response times, it is a good practice to cover at least one or several TCP packet losses by specifying timeouts that are slightly above multiples of 3 seconds (eg: 4 or 5 seconds minimum). If some long-lived sessions are mixed with short-lived sessions (eg: WebSocket and HTTP), it's worth considering "timeout tunnel", which overrides "timeout client" and "timeout server" for tunnels.

This parameter is specific to backends, but can be specified once for all in "defaults" sections. This is in fact one of the easiest solutions not to forget about it. An unspecified timeout results in an infinite timeout, which is not recommended. Such a usage is accepted and works but reports a warning during startup because it may results in accumulation of expired sessions in the system if the system's timeouts are not configured either.

This parameter replaces the old, deprecated "srvtimeout". It is recommended to use it to write new configurations. The form "timeout srvtimeout" is provided only by backwards compatibility but its use is strongly discouraged.

See also : "srvtimeout", "timeout client" and "timeout tunnel".

timeout server-fin <timeout>
Set the inactivity timeout on the server side for half-closed connections.
May be used in sections : defaults | frontend | listen | backend
yes | no | yes | yes
Arguments :
<timeout> is the timeout value specified in milliseconds by default, but can be in any other unit if the number is suffixed by the unit, as explained at the top of this document.

The inactivity timeout applies when the server is expected to acknowledge or send data while one direction is already shut down. This timeout is different from "timeout server" in that it only applies to connections which are closed in one direction. This is particularly useful to avoid keeping connections in FIN_WAIT state for too long when a remote server does not disconnect cleanly. This problem is particularly common long connections such as RDP or WebSocket. Note that this timeout can override "timeout tunnel" when a connection shuts down in one direction. This setting was provided for completeness, but in most situations, it should not be needed.

This parameter is specific to backends, but can be specified once for all in "defaults" sections. By default it is not set, so half-closed connections will use the other timeouts (timeout.server or timeout.tunnel).

See also : "timeout client-fin", "timeout server", and "timeout tunnel".

timeout tarpit <timeout>
Set the duration for which tarpitted connections will be maintained
May be used in sections : defaults | frontend | listen | backend
yes | yes | yes | yes
Arguments :
<timeout> is the tarpit duration specified in milliseconds by default, but can be in any other unit if the number is suffixed by the unit, as explained at the top of this document.

```
8061
8062    When a connection is tarpitted using "reqtarpit", it is maintained open with
8063    no activity for a certain amount of time, then closed. "timeout tarpit"
8064    defines how long it will be maintained open.
8065
8066    The value is specified in milliseconds by default, but can be in any other
8067    unit if the number is suffixed by the unit, as specified at the top of this
8068    document. If unspecified, the same value as the backend's connection timeout
8069    ("timeout connect") is used, for backwards compatibility with older versions
8070    with no "timeout tarpit" parameter.
8071
8072    See also : "timeout connect", "contimeout".
8073
8074
8075  timeout tunnel <timeout>
8076    Set the maximum inactivity time on the client and server side for tunnels.
8077    May be used in sections :   defaults | frontend | listen | backend
8078                                   yes   |    no    |   yes  |   yes
8079    Arguments :
8080      <timeout> is the timeout value specified in milliseconds by default, but
8081                can be in any other unit if the number is suffixed by the unit,
8082                as explained at the top of this document.
8083
8084    The tunnel timeout applies when a bidirectional connection is established
8085    between a client and a server, and the connection remains inactive in both
8086    directions. This timeout supersedes both the client and server timeouts once
8087    the connection becomes a tunnel. In TCP, this timeout is used as soon as no
8088    analyser remains attached to either connection (eg: tcp content rules are
8089    accepted). In HTTP, this timeout is used when a connection is upgraded (eg:
8090    when switching to the WebSocket protocol, or forwarding a CONNECT request
8091    to a proxy), or after the first response when no keepalive/close option is
8092    specified.
8093
8094    Since this timeout is usually used in conjunction with long-lived connections,
8095    it usually is a good idea to also set "timeout client-fin" to handle the
8096    situation where a client suddenly disappears from the net and does not
8097    acknowledge a close, or sends a shutdown and does not acknowledge pending
8098    data anymore. This can happen in lossy networks where firewalls are present,
8099    and is detected by the presence of large amounts of sessions in a FIN_WAIT
8100    state.
8101
8102    The value is specified in milliseconds by default, but can be in any other
8103    unit if the number is suffixed by the unit, as specified at the top of this
8104    document. Whatever the expected normal idle time, it is a good practice to
8105    cover at least one or several TCP packet losses by specifying timeouts that
8106    are slightly above multiples of 3 seconds (eg: 4 or 5 seconds minimum).
8107
8108    This parameter is specific to backends, but can be specified once for all in
8109    "defaults" sections. This is in fact one of the easiest solutions not to
8110    forget about it.
8111
8112    Example :
8113          defaults http
8114              option http-server-close
8115              timeout connect 5s
8116              timeout client 30s
8117              timeout client-fin 30s
8118              timeout server 30s
8119              timeout tunnel  1h    # timeout to use with WebSocket and CONNECT
8120
8121    See also : "timeout client", "timeout client-fin", "timeout server".
8122
8123
8124  transparent (deprecated)
8125    Enable client-side transparent proxying
```



```
8126    May be used in sections :   defaults | frontend | listen | backend
8127                                   yes   |    no    |   yes  |   yes
8128    Arguments : none
8129
8130    This keyword was introduced in order to provide layer 7 persistence to layer
8131    3 load balancers. The idea is to use the OS's ability to redirect an incoming
8132    connection for a remote address to a local process (here HAProxy), and let
8133    this process know what address was initially requested. When this option is
8134    used, sessions without cookies will be forwarded to the original destination
8135    IP address of the incoming request (which should match that of another
8136    equipment), while requests with cookies will still be forwarded to the
8137    appropriate server.
8138
8139    The "transparent" keyword is deprecated, use "option transparent" instead.
8140
8141    Note that contrary to a common belief, this option does NOT make HAProxy
8142    present the client's IP to the server when establishing the connection.
8143
8144    See also: "option transparent"
8145
8146  unique-id-format <string>
8147    Generate a unique ID for each request.
8148    May be used in sections :   defaults | frontend | listen | backend
8149                                   yes   |    yes   |   yes  |   no
8150    Arguments :
8151      <string>   is a log-format string.
8152
8153    This keyword creates a ID for each request using the custom log format. A
8154    unique ID is useful to trace a request passing through many components of
8155    a complex infrastructure. The newly created ID may also be logged using the
8156    %ID tag the log-format string.
8157
8158    The format should be composed from elements that are guaranteed to be
8159    unique when combined together. For instance, if multiple haproxy instances
8160    are involved, it might be important to include the node name. It is often
8161    needed to log the incoming connection's source and destination addresses
8162    and ports. Note that since multiple requests may be performed over the same
8163    connection, including a request counter may help differentiate them.
8164    Similarly, a timestamp may protect against a rollover of the counter.
8165    Logging the process ID will avoid collisions after a service restart.
8166
8167    It is recommended to use hexadecimal notation for many fields since it
8168    makes them more compact and saves space in logs.
8169
8170    Example:
8171
8172          unique-id-format %{+X}o\ %ci:%cp_%fi:%fp_%Ts_%rt:%pid
8173
8174          will generate:
8175
8176                7F000001:8296_7F00001E:1F90_4F7B0A69_0003:790A
8177
8178    See also: "unique-id-header"
8179
8180  unique-id-header <name>
8181    Add a unique ID header in the HTTP request.
8182    May be used in sections :   defaults | frontend | listen | backend
8183                                   yes   |    yes   |   yes  |   no
8184    Arguments :
8185      <name>   is the name of the header.
8186
8187    Add a unique-id header in the HTTP request sent to the server, using the
8188    unique-id-format. It can't work if the unique-id-format doesn't exist.
8189
8190    Example:
```

```
8191
8192          unique-id-format %{+X}o\ %ci:%cp_%fi:%fp_%Ts_%rt:%pid
8193          unique-id-header X-Unique-ID
8194
8195          will generate:
8196
8197             X-Unique-ID: 7F000001:8296_7F00001E:1F90_4F7B0A69_0003:790A
8198
8199       See also: "unique-id-format"
8200
8201    use_backend <backend> [{if | unless} <condition>]
8202      Switch to a specific backend if/unless an ACL-based condition is matched.
8203      May be used in sections :   defaults | frontend | listen | backend
8204                                     no   |    yes   |   yes  |   no
8205      Arguments :
8206        <backend>   is the name of a valid backend or "listen" section, or a
8207                    "log-format" string resolving to a backend name.
8208
8209        <condition> is a condition composed of ACLs, as described in section 7. If
8210                    it is omitted, the rule is unconditionally applied.
8211
8212      When doing content-switching, connections arrive on a frontend and are then
8213      dispatched to various backends depending on a number of conditions. The
8214      relation between the conditions and the backends is described with the
8215      "use_backend" keyword. While it is normally used with HTTP processing, it can
8216      also be used in pure TCP, either without content using stateless ACLs (eg:
8217      source address validation) or combined with a "tcp-request" rule to wait for
8218      some payload.
8219
8220      There may be as many "use_backend" rules as desired. All of these rules are
8221      evaluated in their declaration order, and the first one which matches will
8222      assign the backend.
8223
8224      In the first form, the backend will be used if the condition is met. In the
8225      second form, the backend will be used if the condition is not met. If no
8226      condition is valid, the backend defined with "default_backend" will be used.
8227      If no default backend is defined, either the servers in the same section are
8228      used (in case of a "listen" section) or, in case of a frontend, no server is
8229      used and a 503 service unavailable response is returned.
8230
8231      Note that it is possible to switch from a TCP frontend to an HTTP backend. In
8232      this case, either the frontend has already checked that the protocol is HTTP,
8233      and backend processing will immediately follow, or the backend will wait for
8234      a complete HTTP request to get in. This feature is useful when a frontend
8235      must decode several protocols on a unique port, one of them being HTTP.
8236
8237      When <backend> is a simple name, it is resolved at configuration time, and an
8238      error is reported if the specified backend does not exist. If <backend> is
8239      a log-format string instead, no check may be done at configuration time, so
8240      the backend name is resolved dynamically at run time. If the resulting
8241      backend name does not correspond to any valid backend, no other rule is
8242      evaluated, and the default_backend directive is applied instead. Note that
8243      when using dynamic backend names, it is highly recommended to use a prefix
8244      that no other backend uses in order to ensure that an unauthorized backend
8245      cannot be forced from the request.
8246
8247      It is worth mentioning that "use_backend" rules with an explicit name are
8248      used to detect the association between frontends and backends to compute the
8249      backend's "fullconn" setting. This cannot be done for dynamic names.
8250
8251      See also: "default_backend", "tcp-request", "fullconn", "log-format", and
8252                section 7 about ACLs.
8253
8254
8255    use-server <server> if <condition>
```



```
8256    use-server <server> unless <condition>
8257      Only use a specific server if/unless an ACL-based condition is matched.
8258      May be used in sections :   defaults | frontend | listen | backend
8259                                     no   |    no    |   yes  |   yes
8260      Arguments :
8261        <server>    is the name of a valid server in the same backend section.
8262
8263        <condition> is a condition composed of ACLs, as described in section 7.
8264
8265      By default, connections which arrive to a backend are load-balanced across
8266      the available servers according to the configured algorithm, unless a
8267      persistence mechanism such as a cookie is used and found in the request.
8268
8269      Sometimes it is desirable to forward a particular request to a specific
8270      server without having to declare a dedicated backend for this server. This
8271      can be achieved using the "use-server" rules. These rules are evaluated after
8272      the "redirect" rules and before evaluating cookies, and they have precedence
8273      on them. There may be as many "use-server" rules as desired. All of these
8274      rules are evaluated in their declaration order, and the first one which
8275      matches will assign the server.
8276
8277      If a rule designates a server which is down, and "option persist" is not used
8278      and no force-persist rule was validated, it is ignored and evaluation goes on
8279      with the next rules until one matches.
8280
8281      In the first form, the server will be used if the condition is met. In the
8282      second form, the server will be used if the condition is not met. If no
8283      condition is valid, the processing continues and the server will be assigned
8284      according to other persistence mechanisms.
8285
8286      Note that even if a rule is matched, cookie processing is still performed but
8287      does not assign the server. This allows prefixed cookies to have their prefix
8288      stripped.
8289
8290      The "use-server" statement works both in HTTP and TCP mode. This makes it
8291      suitable for use with content-based inspection. For instance, a server could
8292      be selected in a farm according to the TLS SNI field. And if these servers
8293      have their weight set to zero, they will not be used for other traffic.
8294
8295      Example :
8296         # intercept incoming TLS requests based on the SNI field
8297         use-server www if { req_ssl_sni -i www.example.com }
8298         server     www 192.168.0.1:443 weight 0
8299         use-server mail if { req_ssl_sni -i mail.example.com }
8300         server     mail 192.168.0.1:587 weight 0
8301         use-server imap if { req_ssl_sni -i imap.example.com }
8302         server     mail 192.168.0.1:993 weight 0
8303         # all the rest is forwarded to this server
8304         server  default 192.168.0.2:443 check
8305
8306      See also: "use_backend", section 5 about server and section 7 about ACLs.
8307
8308
8309    5. Bind and Server options
8310    --------------------------
8311
8312    The "bind", "server" and "default-server" keywords support a number of settings
8313    depending on some build options and on the system HAProxy was built on. These
8314    settings generally each consist in one word sometimes followed by a value,
8315    written on the same line as the "bind" or "server" line. All these options are
8316    described in this section.
8317
8318
8319    5.1. Bind options
8320    -----------------
```

```
8321
8322  The "bind" keyword supports a certain number of settings which are all passed
8323  as arguments on the same line. The order in which those arguments appear makes
8324  no importance, provided that they appear after the bind address. All of these
8325  parameters are optional. Some of them consist in a single words (booleans),
8326  while other ones expect a value after them. In this case, the value must be
8327  provided immediately after the setting name.
8328
8329  The currently supported settings are the following ones.
8330
8331  accept-proxy
8332    Enforces the use of the PROXY protocol over any connection accepted by any of
8333    the sockets declared on the same line. Versions 1 and 2 of the PROXY protocol
8334    are supported and correctly detected. The PROXY protocol dictates the layer
8335    3/4 addresses of the incoming connection to be used everywhere an address is
8336    used, with the only exception of "tcp-request connection" rules which will
8337    only see the real connection address. Logs will reflect the addresses
8338    indicated in the protocol, unless it is violated, in which case the real
8339    address will still be used.  This keyword combined with support from external
8340    components can be used as an efficient and reliable alternative to the
8341    X-Forwarded-For mechanism which is not always reliable and not even always
8342    usable. See also "tcp-request connection expect-proxy" for a finer-grained
8343    setting of which client is allowed to use the protocol.
8344
8345  alpn <protocols>
8346    This enables the TLS ALPN extension and advertises the specified protocol
8347    list as supported on top of ALPN. The protocol list consists in a comma-
8348    delimited list of protocol names, for instance: "http/1.1,http/1.0" (without
8349    quotes). This requires that the SSL library is build with support for TLS
8350    extensions enabled (check with haproxy -vv). The ALPN extension replaces the
8351    initial NPN extension.
8352
8353  backlog <backlog>
8354    Sets the socket's backlog to this value. If unspecified, the frontend's
8355    backlog is used instead, which generally defaults to the maxconn value.
8356
8357  ecdhe <named curve>
8358    This setting is only available when support for OpenSSL was built in. It sets
8359    the named curve (RFC 4492) used to generate ECDH ephemeral keys. By default,
8360    used named curve is prime256v1.
8361
8362  ca-file <cafile>
8363    This setting is only available when support for OpenSSL was built in. It
8364    designates a PEM file from which to load CA certificates used to verify
8365    client's certificate.
8366
8367  ca-ignore-err [all|<errorID>,...]
8368    This setting is only available when support for OpenSSL was built in.
8369    Sets a comma separated list of errorIDs to ignore during verify at depth > 0.
8370    If set to 'all', all errors are ignored. SSL handshake is not aborted if an
8371    error is ignored.
8372
8373  ciphers <ciphers>
8374    This setting is only available when support for OpenSSL was built in. It sets
8375    the string describing the list of cipher algorithms ("cipher suite") that are
8376    negotiated during the SSL/TLS handshake. The format of the string is defined
8377    in "man 1 ciphers" from OpenSSL man pages, and can be for instance a string
8378    such as "AES:ALL:!aNULL:!eNULL:+RC4:@STRENGTH" (without quotes).
8379
8380  crl-file <crlfile>
8381    This setting is only available when support for OpenSSL was built in. It
8382    designates a PEM file from which to load certificate revocation list used
8383    to verify client's certificate.
8384
8385  crt <cert>
```



```
8386    This setting is only available when support for OpenSSL was built in. It
8387    designates a PEM file containing both the required certificates and any
8388    associated private keys. This file can be built by concatenating multiple
8389    PEM files into one (e.g. cat cert.pem key.pem > combined.pem). If your CA
8390    requires an intermediate certificate, this can also be concatenated into this
8391    file.
8392
8393    If the OpenSSL used supports Diffie-Hellman, parameters present in this file
8394    are loaded.
8395
8396    If a directory name is used instead of a PEM file, then all files found in
8397    that directory will be loaded in alphabetic order unless their name ends with
8398    '.issuer' or '.ocsp' (reserved extensions). This directive may be specified
8399    multiple times in order to load certificates from multiple files or
8400    directories. The certificates will be presented to clients who provide a valid
8401    TLS Server Name Indication field matching one of their CN or alt subjects.
8402    Wildcards are supported, where a wildcard character '*' is used instead of the
8403    first hostname component (eg: *.example.org matches www.example.org but not
8404    www.sub.example.org).
8405
8406    If no SNI is provided by the client or if the SSL library does not support
8407    TLS extensions, or if the client provides an SNI hostname which does not
8408    match any certificate, then the first loaded certificate will be presented.
8409    This means that when loading certificates from a directory, it is highly
8410    recommended to load the default one first as a file or to ensure that it will
8411    always be the first one in the directory.
8412
8413    Note that the same cert may be loaded multiple times without side effects.
8414
8415    Some CAs (such as Godaddy) offer a drop down list of server types that do not
8416    include HAProxy when obtaining a certificate. If this happens be sure to
8417    choose a webserver that the CA believes requires an intermediate CA (for
8418    Godaddy, selection Apache Tomcat will get the correct bundle, but many
8419    others, e.g. nginx, result in a wrong bundle that will not work for some
8420    clients).
8421
8422    For each PEM file, haproxy checks for the presence of file at the same path
8423    suffixed by ".ocsp". If such file is found, support for the TLS Certificate
8424    Status Request extension (also known as "OCSP stapling") is automatically
8425    enabled. The content of this file is optional. If not empty, it must contain
8426    a valid OCSP Response in DER format. In order to be valid an OCSP Response
8427    must comply with the following rules: it has to indicate a good status,
8428    it has to be a single response for the certificate of the PEM file, and it
8429    has to be valid at the moment of addition. If these rules are not respected
8430    the OCSP Response is ignored and a warning is emitted. In order to  identify
8431    which certificate an OCSP Response applies to, the issuer's certificate is
8432    necessary. If the issuer's certificate is not found in the PEM file, it will
8433    be loaded from a file at the same path as the PEM file suffixed by ".issuer"
8434    if it exists otherwise it will fail with an error.
8435
8436  crt-ignore-err <errors>
8437    This setting is only available when support for OpenSSL was built in. Sets a
8438    comma separated list of errorIDs to ignore during verify at depth == 0.  If
8439    set to 'all', all errors are ignored. SSL handshake is not aborted if an error
8440    is ignored.
8441
8442  crt-list <file>
8443    This setting is only available when support for OpenSSL was built in. It
8444    designates a list of PEM file with an optional list of SNI filter per
8445    certificate, with the following format for each line :
8446
8447          <crtfile> [[!]<snifilter> ...]
8448
8449    Wildcards are supported in the SNI filter. Negative filter are also supported,
8450    only useful in combination with a wildcard filter to exclude a particular SNI.
```

```
8451    The certificates will be presented to clients who provide a valid TLS Server
8452    Name Indication field matching one of the SNI filters. If no SNI filter is
8453    specified, the CN and alt subjects are used. This directive may be specified
8454    multiple times. See the "crt" option for more information. The default
8455    certificate is still needed to meet OpenSSL expectations. If it is not used,
8456    the 'strict-sni' option may be used.
8457
8458  defer-accept
8459    Is an optional keyword which is supported only on certain Linux kernels. It
8460    states that a connection will only be accepted once some data arrive on it,
8461    or at worst after the first retransmit. This should be used only on protocols
8462    for which the client talks first (eg: HTTP). It can slightly improve
8463    performance by ensuring that most of the request is already available when
8464    the connection is accepted. On the other hand, it will not be able to detect
8465    connections which don't talk. It is important to note that this option is
8466    broken in all kernels up to 2.6.31, as the connection is never accepted until
8467    the client talks. This can cause issues with front firewalls which would see
8468    an established connection while the proxy will only see it in SYN_RECV. This
8469    option is only supported on TCPv4/TCPv6 sockets and ignored by other ones.
8470
8471  force-sslv3
8472    This option enforces use of SSLv3 only on SSL connections instantiated from
8473    this listener. SSLv3 is generally less expensive than the TLS counterparts
8474    for high connection rates. This option is also available on global statement
8475    "ssl-default-bind-options". See also "no-tlsv*" and "no-sslv3".
8476
8477  force-tlsv10
8478    This option enforces use of TLSv1.0 only on SSL connections instantiated from
8479    this listener. This option is also available on global statement
8480    "ssl-default-bind-options". See also "no-tlsv*" and "no-sslv3".
8481
8482  force-tlsv11
8483    This option enforces use of TLSv1.1 only on SSL connections instantiated from
8484    this listener. This option is also available on global statement
8485    "ssl-default-bind-options". See also "no-tlsv*", and "no-sslv3".
8486
8487  force-tlsv12
8488    This option enforces use of TLSv1.2 only on SSL connections instantiated from
8489    this listener. This option is also available on global statement
8490    "ssl-default-bind-options". See also "no-tlsv*", and "no-sslv3".
8491
8492  gid <gid>
8493    Sets the group of the UNIX sockets to the designated system gid. It can also
8494    be set by default in the global section's "unix-bind" statement. Note that
8495    some platforms simply ignore this. This setting is equivalent to the "group"
8496    setting except that the group ID is used instead of its name. This setting is
8497    ignored by non UNIX sockets.
8498
8499  group <group>
8500    Sets the group of the UNIX sockets to the designated system group. It can
8501    also be set by default in the global section's "unix-bind" statement. Note
8502    that some platforms simply ignore this. This setting is equivalent to the
8503    "gid" setting except that the group name is used instead of its gid. This
8504    setting is ignored by non UNIX sockets.
8505
8506  id <id>
8507    Fixes the socket ID. By default, socket IDs are automatically assigned, but
8508    sometimes it is more convenient to fix them to ease monitoring. This value
8509    must be strictly positive and unique within the listener/frontend. This
8510    option can only be used when defining only a single socket.
8511
8512  interface <interface>
8513    Restricts the socket to a specific interface. When specified, only packets
8514    received from that particular interface are processed by the socket. This is
8515    currently only supported on Linux. The interface must be a primary system
```



```
8516    interface, not an aliased interface. It is also possible to bind multiple
8517    frontends to the same address if they are bound to different interfaces. Note
8518    that binding to a network interface requires root privileges. This parameter
8519    is only compatible with TCPv4/TCPv6 sockets.
8520
8521  level <level>
8522    This setting is used with the stats sockets only to restrict the nature of
8523    the commands that can be issued on the socket. It is ignored by other
8524    sockets. <level> can be one of :
8525    - "user" is the least privileged level ; only non-sensitive stats can be
8526      read, and no change is allowed. It would make sense on systems where it
8527      is not easy to restrict access to the socket.
8528    - "operator" is the default level and fits most common uses. All data can
8529      be read, and only non-sensitive changes are permitted (eg: clear max
8530      counters).
8531    - "admin" should be used with care, as everything is permitted (eg: clear
8532      all counters).
8533
8534  maxconn <maxconn>
8535    Limits the sockets to this number of concurrent connections. Extraneous
8536    connections will remain in the system's backlog until a connection is
8537    released. If unspecified, the limit will be the same as the frontend's
8538    maxconn. Note that in case of port ranges or multiple addresses, the same
8539    value will be applied to each socket. This setting enables different
8540    limitations on expensive sockets, for instance SSL entries which may easily
8541    eat all memory.
8542
8543  mode <mode>
8544    Sets the octal mode used to define access permissions on the UNIX socket. It
8545    can also be set by default in the global section's "unix-bind" statement.
8546    Note that some platforms simply ignore this. This setting is ignored by non
8547    UNIX sockets.
8548
8549  mss <maxseg>
8550    Sets the TCP Maximum Segment Size (MSS) value to be advertised on incoming
8551    connections. This can be used to force a lower MSS for certain specific
8552    ports, for instance for connections passing through a VPN. Note that this
8553    relies on a kernel feature which is theoretically supported under Linux but
8554    was buggy in all versions prior to 2.6.28. It may or may not work on other
8555    operating systems. It may also not change the advertised value but change the
8556    effective size of outgoing segments. The commonly advertised value for TCPv4
8557    over Ethernet networks is 1460 = 1500(MTU) - 40(IP+TCP). If this value is
8558    positive, it will be used as the advertised MSS. If it is negative, it will
8559    indicate by how much to reduce the incoming connection's advertised MSS for
8560    outgoing segments. This parameter is only compatible with TCP v4/v6 sockets.
8561
8562  name <name>
8563    Sets an optional name for these sockets, which will be reported on the stats
8564    page.
8565
8566  nice <nice>
8567    Sets the 'niceness' of connections initiated from the socket. Value must be
8568    in the range -1024..1024 inclusive, and defaults to zero. Positive values
8569    means that such connections are more friendly to others and easily offer
8570    their place in the scheduler. On the opposite, negative values mean that
8571    connections want to run with a higher priority than others. The difference
8572    only happens under high loads when the system is close to saturation.
8573    Negative values are appropriate for low-latency or administration services,
8574    and high values are generally recommended for CPU intensive tasks such as SSL
8575    processing or bulk transfers which are less sensible to latency. For example,
8576    it may make sense to use a positive value for an SMTP socket and a negative
8577    one for an RDP socket.
8578
8579  no-sslv3
8580    This setting is only available when support for OpenSSL was built in. It
```

```
   disables support for SSLv3 on any sockets instantiated from the listener when
   SSL is supported. Note that SSLv2 is forced disabled in the code and cannot
   be enabled using any configuration option. This option is also available on
   global statement "ssl-default-bind-options". See also "force-tls*",
   and "force-sslv3".

no-tls-tickets
   This setting is only available when support for OpenSSL was built in. It
   disables the stateless session resumption (RFC 5077 TLS Ticket
   extension) and force to use stateful session resumption. Stateless
   session resumption is more expensive in CPU usage. This option is also
   available on global statement "ssl-default-bind-options".

no-tlsv10
   This setting is only available when support for OpenSSL was built in. It
   disables support for TLSv1.0 on any sockets instantiated from the listener
   when SSL is supported. Note that SSLv2 is forced disabled in the code and
   cannot be enabled using any configuration option. This option is also
   available on global statement "ssl-default-bind-options". See also
   "force-tlsv*", and "force-sslv3".

no-tlsv11
   This setting is only available when support for OpenSSL was built in. It
   disables support for TLSv1.1 on any sockets instantiated from the listener
   when SSL is supported. Note that SSLv2 is forced disabled in the code and
   cannot be enabled using any configuration option. This option is also
   available on global statement "ssl-default-bind-options". See also
   "force-tlsv*", and "force-sslv3".

no-tlsv12
   This setting is only available when support for OpenSSL was built in. It
   disables support for TLSv1.2 on any sockets instantiated from the listener
   when SSL is supported. Note that SSLv2 is forced disabled in the code and
   cannot be enabled using any configuration option. This option is also
   available on global statement "ssl-default-bind-options". See also
   "force-tlsv*", and "force-sslv3".

npn <protocols>
   This enables the NPN TLS extension and advertises the specified protocol list
   as supported on top of NPN. The protocol list consists in a comma-delimited
   list of protocol names, for instance: "http/1.1,http/1.0" (without quotes).
   This requires that the SSL library is build with support for TLS extensions
   enabled (check with haproxy -vv). Note that the NPN extension has been
   replaced with the ALPN extension (see the "alpn" keyword).

process [ all | odd | even | <number 1-64>[-<number 1-64>] ]
   This restricts the list of processes on which this listener is allowed to
   run. It does not enforce any process but eliminates those which do not match.
   If the frontend uses a "bind-process" setting, the intersection between the
   two is applied. If in the end the listener is not allowed to run on any
   remaining process, a warning is emitted, and the listener will either run on
   the first process of the listener if a single process was specified, or on
   all of its processes if multiple processes were specified. For the unlikely
   case where several ranges are needed, this directive may be repeated. The
   main purpose of this directive is to be used with the stats sockets and have
   one different socket per process. The second purpose is to have multiple bind
   lines sharing the same IP:port but not the same process in a listener, so
   that the system can distribute the incoming connections into multiple queues
   and allow a smoother inter-process load balancing. Currently Linux 3.9 and
   above is known for supporting this. See also "bind-process" and "nbproc".

ssl
   This setting is only available when support for OpenSSL was built in. It
   enables SSL deciphering on connections instantiated from this listener. A
   certificate is necessary (see "crt" above). All contents in the buffers will
```



```
   appear in clear text, so that ACLs and HTTP processing will only have access
   to deciphered contents.

strict-sni
   This setting is only available when support for OpenSSL was built in. The
   SSL/TLS negotiation is allow only if the client provided an SNI which match
   a certificate. The default certificate is not used.
   See the "crt" option for more information.

tfo
   Is an optional keyword which is supported only on Linux kernels >= 3.7. It
   enables TCP Fast Open on the listening socket, which means that clients which
   support this feature will be able to send a request and receive a response
   during the 3-way handshake starting from second connection, thus saving one
   round-trip after the first connection. This only makes sense with protocols
   that use high connection rates and where each round trip matters. This can
   possibly cause issues with many firewalls which do not accept data on SYN
   packets, so this option should only be enabled once well tested. This option
   is only supported on TCPv4/TCPv6 sockets and ignored by other ones. You may
   need to build HAProxy with USE_TFO=1 if your libc doesn't define
   TCP_FASTOPEN.

transparent
   Is an optional keyword which is supported only on certain Linux kernels. It
   indicates that the addresses will be bound even if they do not belong to the
   local machine, and that packets targeting any of these addresses will be
   intercepted just as if the addresses were locally configured. This normally
   requires that IP forwarding is enabled. Caution! do not use this with the
   default address '*', as it would redirect any traffic for the specified port.
   This keyword is available only when HAProxy is built with USE_LINUX_TPROXY=1.
   This parameter is only compatible with TCPv4 and TCPv6 sockets, depending on
   kernel version. Some distribution kernels include backports of the feature,
   so check for support with your vendor.

v4v6
   Is an optional keyword which is supported only on most recent systems
   including Linux kernels >= 2.4.21. It is used to bind a socket to both IPv4
   and IPv6 when it uses the default address. Doing so is sometimes necessary
   on systems which bind to IPv6 only by default. It has no effect on non-IPv6
   sockets, and is overridden by the "v6only" option.

v6only
   Is an optional keyword which is supported only on most recent systems
   including Linux kernels >= 2.4.21. It is used to bind a socket to IPv6 only
   when it uses the default address. Doing so is sometimes preferred to doing it
   system-wide as it is per-listener. It has no effect on non-IPv6 sockets and
   has precedence over the "v4v6" option.

uid <uid>
   Sets the owner of the UNIX sockets to the designated system uid. It can also
   be set by default in the global section's "unix-bind" statement. Note that
   some platforms simply ignore this. This setting is equivalent to the "user"
   setting except that the user numeric ID is used instead of its name. This
   setting is ignored by non UNIX sockets.

user <user>
   Sets the owner of the UNIX sockets to the designated system user. It can also
   be set by default in the global section's "unix-bind" statement. Note that
   some platforms simply ignore this. This setting is equivalent to the "uid"
   setting except that the user name is used instead of its uid. This setting is
   ignored by non UNIX sockets.

verify [none|optional|required]
   This setting is only available when support for OpenSSL was built in. If set
   to 'none', client certificate is not requested. This is the default. In other
```

cases, a client certificate is requested. If the client does not provide a certificate after the request and if 'verify' is set to 'required', then the handshake is aborted, while it would have succeeded if set to 'optional'. The certificate provided by the client is always verified using CAs from 'ca-file' and optional CRLs from 'crl-file'. On verify failure the handshake is aborted, regardless of the 'verify' option, unless the error code exactly matches one of those listed with 'ca-ignore-err' or 'crt-ignore-err'.

5.2. Server and default-server options

The "server" and "default-server" keywords support a certain number of settings which are all passed as arguments on the server line. The order in which those arguments appear does not count, and they are all optional. Some of those settings are single words (booleans) while others expect one or several values after them. In this case, the values must immediately follow the setting name. Except default-server, all those settings must be specified after the server's address if they are used:

```
server <name> <address>[:port] [settings ...]
default-server [settings ...]
```

The currently supported settings are the following ones.

addr <ipv4|ipv6>
  Using the "addr" parameter, it becomes possible to use a different IP address to send health-checks. On some servers, it may be desirable to dedicate an IP address to specific component able to perform complex tests which are more suitable to health-checks than the application. This parameter is ignored if the "check" parameter is not set. See also the "port" parameter.

  Supported in default-server: No

agent-check
  Enable an auxiliary agent check which is run independently of a regular health check. An agent health check is performed by making a TCP connection to the port set by the "agent-port" parameter and reading an ASCII string. The string is made of a series of words delimited by spaces, tabs or commas in any order, optionally terminated by '\r' and/or '\n', each consisting of :

  - An ASCII representation of a positive integer percentage, e.g. "75%". Values in this format will set the weight proportional to the initial weight of a server as configured when haproxy starts. Note that a zero weight is reported on the stats page as "DRAIN" since it has the same effect on the server (it's removed from the LB farm).

  - The word "ready". This will turn the server's administrative state to the READY mode, thus cancelling any DRAIN or MAINT state

  - The word "drain". This will turn the server's administrative state to the DRAIN mode, thus it will not accept any new connections other than those that are accepted via persistence.

  - The word "maint". This will turn the server's administrative state to the MAINT mode, thus it will not accept any new connections at all, and health checks will be stopped.

  - The words "down", "failed", or "stopped", optionally followed by a description string after a sharp ('#'). All of these mark the server's operating state as DOWN, but since the word itself is reported on the stats page, the difference allows an administrator to know if the situation was expected or not : the service may intentionally be stopped, may appear up but fail some validity tests, or may be seen as down (eg: missing process, or port not responding).

  - The word "up" sets back the server's operating state as UP if health checks also report that the service is accessible.

  Parameters which are not advertised by the agent are not changed. For example, an agent might be designed to monitor CPU usage and only report a relative weight and never interact with the operating status. Similarly, an agent could be designed as an end-user interface with 3 radio buttons allowing an administrator to change only the administrative state. However, it is important to consider that only the agent may revert its own actions, so if a server is set to DRAIN mode or to DOWN state using the agent, the agent must implement the other equivalent actions to bring the service into operations again.

  Failure to connect to the agent is not considered an error as connectivity is tested by the regular health check which is enabled by the "check" parameter. Warning though, it is not a good idea to stop an agent after it reports "down", since only an agent reporting "up" will be able to turn the server up again. Note that the CLI on the Unix stats socket is also able to force an agent's result in order to workaround a bogus agent if needed.

  Requires the "agent-port" parameter to be set. See also the "agent-inter" parameter.

  Supported in default-server: No

agent-inter <delay>
  The "agent-inter" parameter sets the interval between two agent checks to <delay> milliseconds. If left unspecified, the delay defaults to 2000 ms.

  Just as with every other time-based parameter, it may be entered in any other explicit unit among { us, ms, s, m, h, d }. The "agent-inter" parameter also serves as a timeout for agent checks "timeout check" is not set. In order to reduce "resonance" effects when multiple servers are hosted on the same hardware, the agent and health checks of all servers are started with a small time offset between them. It is also possible to add some random noise in the agent and health checks interval using the global "spread-checks" keyword. This makes sense for instance when a lot of backends use the same servers.

  See also the "agent-check" and "agent-port" parameters.

  Supported in default-server: Yes

agent-port <port>
  The "agent-port" parameter sets the TCP port used for agent checks.

  See also the "agent-check" and "agent-inter" parameters.

  Supported in default-server: Yes

backup
  When "backup" is present on a server line, the server is only used in load balancing when all other non-backup servers are unavailable. Requests coming with a persistence cookie referencing the server will always be served though. By default, only the first operational backup server is used, unless the "allbackups" option is set in the backend. See also the "allbackups" option.

  Supported in default-server: No

ca-file <cafile>
  This setting is only available when support for OpenSSL was built in. It designates a PEM file from which to load CA certificates used to verify server's certificate.

```
8841    Supported in default-server: No
8842
8843  check
8844    This option enables health checks on the server. By default, a server is
8845    always considered available. If "check" is set, the server is available when
8846    accepting periodic TCP connections, to ensure that it is really able to serve
8847    requests. The default address and port to send the tests to are those of the
8848    server, and the default source is the same as the one defined in the
8849    backend. It is possible to change the address using the "addr" parameter, the
8850    port using the "port" parameter, the source address using the "source"
8851    address, and the interval and timers using the "inter", "rise" and "fall"
8852    parameters. The request method is define in the backend using the "httpchk",
8853    "smtpchk", "mysql-check", "pgsql-check" and "ssl-hello-chk" options. Please
8854    refer to those options and parameters for more information.
8855
8856    Supported in default-server: No
8857
8858  check-send-proxy
8859    This option forces emission of a PROXY protocol line with outgoing health
8860    checks, regardless of whether the server uses send-proxy or not for the
8861    normal traffic. By default, the PROXY protocol is enabled for health checks
8862    if it is already enabled for normal traffic and if no "port" nor "addr"
8863    directive is present. However, if such a directive is present, the
8864    "check-send-proxy" option needs to be used to force the use of the
8865    protocol. See also the "send-proxy" option for more information.
8866
8867    Supported in default-server: No
8868
8869  check-ssl
8870    This option forces encryption of all health checks over SSL, regardless of
8871    whether the server uses SSL or not for the normal traffic. This is generally
8872    used when an explicit "port" or "addr" directive is specified and SSL health
8873    checks are not inherited. It is important to understand that this option
8874    inserts an SSL transport layer below the checks, so that a simple TCP connect
8875    check becomes an SSL connect, which replaces the old ssl-hello-chk. The most
8876    common use is to send HTTPS checks by combining "httpchk" with SSL checks.
8877    All SSL settings are common to health checks and traffic (eg: ciphers).
8878    See the "ssl" option for more information.
8879
8880    Supported in default-server: No
8881
8882  ciphers <ciphers>
8883    This option sets the string describing the list of cipher algorithms that is
8884    is negotiated during the SSL/TLS handshake with the server. The format of the
8885    string is defined in "man 1 ciphers". When SSL is used to communicate with
8886    servers on the local network, it is common to see a weaker set of algorithms
8887    than what is used over the internet. Doing so reduces CPU usage on both the
8888    server and haproxy while still keeping it compatible with deployed software.
8889    Some algorithms such as RC4-SHA1 are reasonably cheap. If no security at all
8890    is needed and just connectivity, using DES can be appropriate.
8891
8892    Supported in default-server: No
8893
8894  cookie <value>
8895    The "cookie" parameter sets the cookie value assigned to the server to
8896    <value>. This value will be checked in incoming requests, and the first
8897    operational server possessing the same value will be selected. In return, in
8898    cookie insertion or rewrite modes, this value will be assigned to the cookie
8899    sent to the client. There is nothing wrong in having several servers sharing
8900    the same cookie value, and it is in fact somewhat common between normal and
8901    backup servers. See also the "cookie" keyword in backend section.
8902
8903    Supported in default-server: No
8904
8905  crl-file <crlfile>
```



```
8906    This setting is only available when support for OpenSSL was built in. It
8907    designates a PEM file from which to load certificate revocation list used
8908    to verify server's certificate.
8909
8910    Supported in default-server: No
8911
8912  crt <cert>
8913    This setting is only available when support for OpenSSL was built in.
8914    It designates a PEM file from which to load both a certificate and the
8915    associated private key. This file can be built by concatenating both PEM
8916    files into one. This certificate will be sent if the server send a client
8917    certificate request.
8918
8919    Supported in default-server: No
8920
8921  disabled
8922    The "disabled" keyword starts the server in the "disabled" state. That means
8923    that it is marked down in maintenance mode, and no connection other than the
8924    ones allowed by persist mode will reach it. It is very well suited to setup
8925    new servers, because normal traffic will never reach them, while it is still
8926    possible to test the service by making use of the force-persist mechanism.
8927
8928    Supported in default-server: No
8929
8930  error-limit <count>
8931    If health observing is enabled, the "error-limit" parameter specifies the
8932    number of consecutive errors that triggers event selected by the "on-error"
8933    option. By default it is set to 10 consecutive errors.
8934
8935    Supported in default-server: Yes
8936
8937    See also the "check", "error-limit" and "on-error".
8938
8939  fall <count>
8940    The "fall" parameter states that a server will be considered as dead after
8941    <count> consecutive unsuccessful health checks. This value defaults to 3 if
8942    unspecified. See also the "check", "inter" and "rise" parameters.
8943
8944    Supported in default-server: Yes
8945
8946  force-sslv3
8947    This option enforces use of SSLv3 only when SSL is used to communicate with
8948    the server. SSLv3 is generally less expensive than the TLS counterparts for
8949    high connection rates. This option is also available on global statement
8950    "ssl-default-server-options". See also "no-tlsv*", "no-sslv3".
8951
8952    Supported in default-server: No
8953
8954  force-tlsv10
8955    This option enforces use of TLSv1.0 only when SSL is used to communicate with
8956    the server. This option is also available on global statement
8957    "ssl-default-server-options". See also "no-tlsv*", "no-sslv3".
8958
8959    Supported in default-server: No
8960
8961  force-tlsv11
8962    This option enforces use of TLSv1.1 only when SSL is used to communicate with
8963    the server. This option is also available on global statement
8964    "ssl-default-server-options". See also "no-tlsv*", "no-sslv3".
8965
8966    Supported in default-server: No
8967
8968  force-tlsv12
8969    This option enforces use of TLSv1.2 only when SSL is used to communicate with
8970    the server. This option is also available on global statement
```

"ssl-default-server-options". See also "no-tlsv*", "no-sslv3".

Supported in default-server: No

id <value>
Set a persistent ID for the server. This ID must be positive and unique for the proxy. An unused ID will automatically be assigned if unset. The first assigned value will be 1. This ID is currently only returned in statistics.

Supported in default-server: No

inter <delay>
fastinter <delay>
downinter <delay>
The "inter" parameter sets the interval between two consecutive health checks to <delay> milliseconds. If left unspecified, the delay defaults to 2000 ms. It is also possible to use "fastinter" and "downinter" to optimize delays between checks depending on the server state :

Server state	Interval used
UP 100% (non-transitional)	"inter"
Transitionally UP (going down), Transitionally DOWN (going up), or yet unchecked.	"fastinter" if set, "inter" otherwise.
DOWN 100% (non-transitional)	"downinter" if set, "inter" otherwise.

Just as with every other time-based parameter, they can be entered in any other explicit unit among { us, ms, s, m, h, d }. The "inter" parameter also serves as a timeout for health checks sent to servers if "timeout check" is not set. In order to reduce "resonance" effects when multiple servers are hosted on the same hardware, the agent and health checks of all servers are started with a small time offset between them. It is also possible to add some random noise in the agent and health checks interval using the global "spread-checks" keyword. This makes sense for instance when a lot of backends use the same servers.

Supported in default-server: Yes

maxconn <maxconn>
The "maxconn" parameter specifies the maximal number of concurrent connections that will be sent to this server. If the number of incoming concurrent requests goes higher than this value, they will be queued, waiting for a connection to be released. This parameter is very important as it can save fragile servers from going down under extreme loads. If a "minconn" parameter is specified, the limit becomes dynamic. The default value is "0" which means unlimited. See also the "minconn" and "maxqueue" parameters, and the backend's "fullconn" keyword.

Supported in default-server: Yes

maxqueue <maxqueue>
The "maxqueue" parameter specifies the maximal number of connections which will wait in the queue for this server. If this limit is reached, next requests will be redispatched to other servers instead of indefinitely waiting to be served. This will break persistence but may allow people to quickly re-log in when the server they try to connect to is dying. The default value is "0" which means the queue is unlimited. See also the "maxconn" and "minconn" parameters.

Supported in default-server: Yes

minconn <minconn>
When the "minconn" parameter is set, the maxconn limit becomes a dynamic limit following the backend's load. The server will always accept at least <minconn> connections, never more than <maxconn>, and the limit will be on the ramp between both values when the backend has less than <fullconn> concurrent connections. This makes it possible to limit the load on the server during normal loads, but push it further for important loads without overloading the server during exceptional loads. See also the "maxconn" and "maxqueue" parameters, as well as the "fullconn" backend keyword.

Supported in default-server: Yes

no-sslv3
This option disables support for SSLv3 when SSL is used to communicate with the server. Note that SSLv2 is disabled in the code and cannot be enabled using any configuration option. See also "force-sslv3", "force-tlsv*".

Supported in default-server: No

no-tls-tickets
This setting is only available when support for OpenSSL was built in. It disables the stateless session resumption (RFC 5077 TLS Ticket extension) and force to use stateful session resumption. Stateless session resumption is more expensive in CPU usage for servers. This option is also available on global statement "ssl-default-server-options".

Supported in default-server: No

no-tlsv10
This option disables support for TLSv1.0 when SSL is used to communicate with the server. Note that SSLv2 is disabled in the code and cannot be enabled using any configuration option. TLSv1 is more expensive than SSLv3 so it often makes sense to disable it when communicating with local servers. This option is also available on global statement "ssl-default-server-options". See also "force-sslv3", "force-tlsv*".

Supported in default-server: No

no-tlsv11
This option disables support for TLSv1.1 when SSL is used to communicate with the server. Note that SSLv2 is disabled in the code and cannot be enabled using any configuration option. TLSv1 is more expensive than SSLv3 so it often makes sense to disable it when communicating with local servers. This option is also available on global statement "ssl-default-server-options". See also "force-sslv3", "force-tlsv*".

Supported in default-server: No

no-tlsv12
This option disables support for TLSv1.2 when SSL is used to communicate with the server. Note that SSLv2 is disabled in the code and cannot be enabled using any configuration option. TLSv1 is more expensive than SSLv3 so it often makes sense to disable it when communicating with local servers. This option is also available on global statement "ssl-default-server-options". See also "force-sslv3", "force-tlsv*".

Supported in default-server: No

non-stick
Never add connections allocated to this sever to a stick-table. This may be used in conjunction with backup to ensure that stick-table persistence is disabled for backup servers.

Supported in default-server: No

```
observe <mode>
  This option enables health adjusting based on observing communication with
  the server. By default this functionality is disabled and enabling it also
  requires to enable health checks. There are two supported modes: "layer4" and
  "layer7". In layer4 mode, only successful/unsuccessful tcp connections are
  significant. In layer7, which is only allowed for http proxies, responses
  received from server are verified, like valid/wrong http code, unparsable
  headers, a timeout, etc. Valid status codes include 100 to 499, 501 and 505.

  Supported in default-server: No

  See also the "check", "on-error" and "error-limit".

on-error <mode>
  Select what should happen when enough consecutive errors are detected.
  Currently, four modes are available:
  - fastinter: force fastinter
  - fail-check: simulate a failed check, also forces fastinter (default)
  - sudden-death: simulate a pre-fatal failed health check, one more failed
    check will mark a server down, forces fastinter
  - mark-down: mark the server immediately down and force fastinter

  Supported in default-server: Yes

  See also the "check", "observe" and "error-limit".

on-marked-down <action>
  Modify what occurs when a server is marked down.
  Currently one action is available:
  - shutdown-sessions: Shutdown peer sessions. When this setting is enabled,
    all connections to the server are immediately terminated when the server
    goes down. It might be used if the health check detects more complex cases
    than a simple connection status, and long timeouts would cause the service
    to remain unresponsive for too long a time. For instance, a health check
    might detect that a database is stuck and that there's no chance to reuse
    existing connections anymore. Connections killed this way are logged with
    a 'D' termination code (for "Down").

  Actions are disabled by default

  Supported in default-server: Yes

on-marked-up <action>
  Modify what occurs when a server is marked up.
  Currently one action is available:
  - shutdown-backup-sessions: Shutdown sessions on all backup servers. This is
    done only if the server is not in backup state and if it is not disabled
    (it must have an effective weight > 0). This can be used sometimes to force
    an active server to take all the traffic back after recovery when dealing
    with long sessions (eg: LDAP, SQL, ...). Doing this can cause more trouble
    than it tries to solve (eg: incomplete transactions), so use this feature
    with extreme care. Sessions killed because a server comes up are logged
    with an 'U' termination code (for "Up").

  Actions are disabled by default

  Supported in default-server: Yes

port <port>
  Using the "port" parameter, it becomes possible to use a different port to
  send health-checks. On some servers, it may be desirable to dedicate a port
  to a specific component able to perform complex tests which are more suitable
  to health-checks than the application. It is common to run a simple script in
  inetd for instance. This parameter is ignored if the "check" parameter is not
  set. See also the "addr" parameter.
```



```

  Supported in default-server: Yes

redir <prefix>
  The "redir" parameter enables the redirection mode for all GET and HEAD
  requests addressing this server. This means that instead of having HAProxy
  forward the request to the server, it will send an "HTTP 302" response with
  the "Location" header composed of this prefix immediately followed by the
  requested URI beginning at the leading '/' of the path component. That means
  that no trailing slash should be used after <prefix>. All invalid requests
  will be rejected, and all non-GET or HEAD requests will be normally served by
  the server. Note that since the response is completely forged, no header
  mangling nor cookie insertion is possible in the response. However, cookies in
  requests are still analysed, making this solution completely usable to direct
  users to a remote location in case of local disaster. Main use consists in
  increasing bandwidth for static servers by having the clients directly
  connect to them. Note: never use a relative location here, it would cause a
  loop between the client and HAProxy!

  Example :  server srv1 192.168.1.1:80 redir http://image1.mydomain.com check

  Supported in default-server: No

rise <count>
  The "rise" parameter states that a server will be considered as operational
  after <count> consecutive successful health checks. This value defaults to 2
  if unspecified. See also the "check", "inter" and "fall" parameters.

  Supported in default-server: Yes

send-proxy
  The "send-proxy" parameter enforces use of the PROXY protocol over any
  connection established to this server. The PROXY protocol informs the other
  end about the layer 3/4 addresses of the incoming connection, so that it can
  know the client's address or the public address it accessed to, whatever the
  upper layer protocol. For connections accepted by an "accept-proxy" listener,
  the advertised address will be used. Only TCPv4 and TCPv6 address families
  are supported. Other families such as Unix sockets, will report an UNKNOWN
  family. Servers using this option can fully be chained to another instance of
  haproxy listening with an "accept-proxy" setting. This setting must not be
  used if the server isn't aware of the protocol. When health checks are sent
  to the server, the PROXY protocol is automatically used when this option is
  set, unless there is an explicit "port" or "addr" directive, in which case an
  explicit "check-send-proxy" directive would also be needed to use the PROXY
  protocol. See also the "accept-proxy" option of the "bind" keyword.

  Supported in default-server: No

send-proxy-v2
  The "send-proxy-v2" parameter enforces use of the PROXY protocol version 2
  over any connection established to this server. The PROXY protocol informs
  the other end about the layer 3/4 addresses of the incoming connection, so
  that it can know the client's address or the public address it accessed to,
  whatever the upper layer protocol. This setting must not be used if the
  server isn't aware of this version of the protocol. See also the "send-proxy"
  option of the "bind" keyword.

  Supported in default-server: No

send-proxy-v2-ssl
  The "send-proxy-v2-ssl" parameter enforces use of the PROXY protocol version
  2 over any connection established to this server. The PROXY protocol informs
  the other end about the layer 3/4 addresses of the incoming connection, so
  that it can know the client's address or the public address it accessed to,
  whatever the upper layer protocol. In addition, the SSL information extension
```

```
   of the PROXY protocol is added to the PROXY protocol header. This setting
   must not be used if the server isn't aware of this version of the protocol.
   See also the "send-proxy-v2" option of the "bind" keyword.

   Supported in default-server: No

send-proxy-v2-ssl-cn
   The "send-proxy-v2-ssl" parameter enforces use of the PROXY protocol version
   2 over any connection established to this server. The PROXY protocol informs
   the other end about the layer 3/4 addresses of the incoming connection, so
   that it can know the client's address or the public address it accessed to,
   whatever the upper layer protocol. In addition, the SSL information extension
   of the PROXY protocol, along along with the Common Name from the subject of
   the client certificate (if any), is added to the PROXY protocol header. This
   setting must not be used if the server isn't aware of this version of the
   protocol. See also the "send-proxy-v2" option of the "bind" keyword.

   Supported in default-server: No

slowstart <start_time_in_ms>
   The "slowstart" parameter for a server accepts a value in milliseconds which
   indicates after how long a server which has just come back up will run at
   full speed. Just as with every other time-based parameter, it can be entered
   in any other explicit unit among { us, ms, s, m, h, d }. The speed grows
   linearly from 0 to 100% during this time. The limitation applies to two
   parameters :

   - maxconn: the number of connections accepted by the server will grow from 1
     to 100% of the usual dynamic limit defined by (minconn,maxconn,fullconn).

   - weight: when the backend uses a dynamic weighted algorithm, the weight
     grows linearly from 1 to 100%. In this case, the weight is updated at every
     health-check. For this reason, it is important that the "inter" parameter
     is smaller than the "slowstart", in order to maximize the number of steps.

   The slowstart never applies when haproxy starts, otherwise it would cause
   trouble to running servers. It only applies when a server has been previously
   seen as failed.

   Supported in default-server: Yes

source <addr>[:<pl>[-<ph>]] [usesrc { <addr2>[:<port2>] | client | clientip } ]
source <addr>[:<port>] [usesrc { <addr2>[:<port2>] | hdr_ip(<hdr>[,<occ>]) } ]
source <addr>[:<pl>[-<ph>]] [interface <name>] ...
   The "source" parameter sets the source address which will be used when
   connecting to the server. It follows the exact same parameters and principle
   as the backend "source" keyword, except that it only applies to the server
   referencing it. Please consult the "source" keyword for details.

   Additionally, the "source" statement on a server line allows one to specify a
   source port range by indicating the lower and higher bounds delimited by a
   dash ('-'). Some operating systems might require a valid IP address when a
   source port range is specified. It is permitted to have the same IP/range for
   several servers. Doing so makes it possible to bypass the maximum of 64k
   total concurrent connections. The limit will then reach 64k connections per
   server.

   Supported in default-server: No

ssl
   This option enables SSL ciphering on outgoing connections to the server. It
   is critical to verify server certificates using "verify" when using SSL to
   connect to servers, otherwise the communication is prone to trivial man in
   the-middle attacks rendering SSL useless. When this option is used, health
   checks are automatically sent in SSL too unless there is a "port" or an
```



```
   "addr" directive indicating the check should be sent to a different location.
   See the "check-ssl" option to force SSL health checks.

   Supported in default-server: No

track [<proxy>/]<server>
   This option enables ability to set the current state of the server by tracking
   another one. It is possible to track a server which itself tracks another
   server, provided that at the end of the chain, a server has health checks
   enabled. If <proxy> is omitted the current one is used. If disable-on-404 is
   used, it has to be enabled on both proxies.

   Supported in default-server: No

verify [none|required]
   This setting is only available when support for OpenSSL was built in. If set
   to 'none', server certificate is not verified. In the other case, The
   certificate provided by the server is verified using CAs from 'ca-file'
   and optional CRLs from 'crl-file'. If 'ssl_server_verify' is not specified
   in global  section, this is the default. On verify failure the handshake
   is aborted. It is critically important to verify server certificates when
   using SSL to connect to servers, otherwise the communication is prone to
   trivial man-in-the-middle attacks rendering SSL totally useless.

   Supported in default-server: No

verifyhost <hostname>
   This setting is only available when support for OpenSSL was built in, and
   only takes effect if 'verify required' is also specified. When set, the
   hostnames in the subject and subjectAlternateNames of the certificate
   provided by the server are checked. If none of the hostnames in the
   certificate match the specified hostname, the handshake is aborted. The
   hostnames in the server-provided certificate may include wildcards.

   Supported in default-server: No

weight <weight>
   The "weight" parameter is used to adjust the server's weight relative to
   other servers. All servers will receive a load proportional to their weight
   relative to the sum of all weights, so the higher the weight, the higher the
   load. The default weight is 1, and the maximal value is 256. A value of 0
   means the server will not participate in load-balancing but will still accept
   persistent connections. If this parameter is used to distribute the load
   according to server's capacity, it is recommended to start with values which
   can both grow and shrink, for instance between 10 and 100 to leave enough
   room above and below for later adjustments.

   Supported in default-server: Yes


6. HTTP header manipulation
---------------------------

In HTTP mode, it is possible to rewrite, add or delete some of the request and
response headers based on regular expressions. It is also possible to block a
request or a response if a particular header matches a regular expression,
which is enough to stop most elementary protocol attacks, and to protect
against information leak from the internal network.

If HAProxy encounters an "Informational Response" (status code 1xx), it is able
to process all rsp* rules which can allow, deny, rewrite or delete a header,
but it will refuse to add a header to any such messages as this is not
HTTP-compliant. The reason for still processing headers in such responses is to
stop and/or fix any possible information leak which may happen, for instance
because another downstream equipment would unconditionally add a header, or if
```

a server name appears there. When such messages are seen, normal processing still occurs on the next non-informational messages.

This section covers common usage of the following keywords, described in detail in section 4.2 :

  - reqadd   <string>
  - reqallow   <search>
  - reqiallow <search>
  - reqdel   <search>
  - reqidel   <search>
  - reqdeny   <search>
  - reqideny   <search>
  - reqpass   <search>
  - reqipass   <search>
  - reqrep   <search> <replace>
  - reqirep   <search> <replace>
  - reqtarpit <search>
  - reqitarpit <search>
  - rspadd   <string>
  - rspdel   <search>
  - rspidel   <search>
  - rspdeny   <search>
  - rspideny   <search>
  - rsprep   <search> <replace>
  - rspirep   <search> <replace>

With all these keywords, the same conventions are used. The <search> parameter is a POSIX extended regular expression (regex) which supports grouping through parenthesis (without the backslash). Spaces and other delimiters must be prefixed with a backslash ('\') to avoid confusion with a field delimiter. Other characters may be prefixed with a backslash to change their meaning :

  \t   for a tab
  \r   for a carriage return (CR)
  \n   for a new line (LF)
  \   to mark a space and differentiate it from a delimiter
  \#   to mark a sharp and differentiate it from a comment
  \\   to use a backslash in a regex
  \\\\ to use a backslash in the text (*2 for regex, *2 for haproxy)
  \xXX to write the ASCII hex code XX as in the C language

The <replace> parameter contains the string to be used to replace the largest portion of text matching the regex. It can make use of the special characters above, and can reference a substring which is delimited by parenthesis in the regex, by writing a backslash ('\') immediately followed by one digit from 0 to 9 indicating the group position (0 designating the entire line). This practice is very common to users of the "sed" program.

The <string> parameter represents the string which will systematically be added after the last header line. It can also use special character sequences above.

Notes related to these keywords :

  - these keywords are not always convenient to allow/deny based on header
    contents. It is strongly recommended to use ACLs with the "block" keyword
    instead, resulting in far more flexible and manageable rules.

  - lines are always considered as a whole. It is not possible to reference
    a header name only or a value only. This is important because of the way
    headers are written (notably the number of spaces after the colon).

  - the first line is always considered as a header, which makes it possible to
    rewrite or filter HTTP requests URIs or response codes, but in turn makes
    it harder to distinguish between headers and request line. The regex prefix

    ^[^\ \t]*[\ \t] matches any HTTP method followed by a space, and the prefix
    ^[^ \t:]*: matches any header name followed by a colon.

  - for performances reasons, the number of characters added to a request or to
    a response is limited at build time to values between 1 and 4 kB. This
    should normally be far more than enough for most usages. If it is too short
    on occasional usages, it is possible to gain some space by removing some
    useless headers before adding new ones.

  - keywords beginning with "reqi" and "rspi" are the same as their counterpart
    without the 'i' letter except that they ignore case when matching patterns.

  - when a request passes through a frontend then a backend, all req* rules
    from the frontend will be evaluated, then all req* rules from the backend
    will be evaluated. The reverse path is applied to responses.

  - req* statements are applied after "block" statements, so that "block" is
    always the first one, but before "use_backend" in order to permit rewriting
    before switching.

7. Using ACLs and fetching samples

Haproxy is capable of extracting data from request or response streams, from client or server information, from tables, environmental information etc... The action of extracting such data is called fetching a sample. Once retrieved, these samples may be used for various purposes such as a key to a stick-table, but most common usages consist in matching them against predefined constant data called patterns.

7.1. ACL basics

The use of Access Control Lists (ACL) provides a flexible solution to perform content switching and generally to take decisions based on content extracted from the request, the response or any environmental status. The principle is simple :

  - extract a data sample from a stream, table or the environment
  - optionally apply some format conversion to the extracted sample
  - apply one or multiple pattern matching methods on this sample
  - perform actions only when a pattern matches the sample

The actions generally consist in blocking a request, selecting a backend, or adding a header.

In order to define a test, the "acl" keyword is used. The syntax is :

   acl <aclname> <criterion> [flags] [operator] [<value>] ...

This creates a new ACL <aclname> or completes an existing one with new tests. Those tests apply to the portion of request/response specified in <criterion> and may be adjusted with optional flags [flags]. Some criteria also support an operator which may be specified before the set of values. Optionally some conversion operators may be applied to the sample, and they will be specified as a comma-delimited list of keywords just after the first keyword. The values are of the type supported by the criterion, and are separated by spaces.

ACL names must be formed from upper and lower case letters, digits, '-' (dash), '_' (underscore) , '.' (dot) and ':' (colon). ACL names are case-sensitive, which means that "my_acl" and "My_Acl" are two different ACLs.

There is no enforced limit to the number of ACLs. The unused ones do not affect

performance, they just consume a small amount of memory.

The criterion generally is the name of a sample fetch method, or one of its ACL specific declinations. The default test method is implied by the output type of this sample fetch method. The ACL declinations can describe alternate matching methods of a same sample fetch method. The sample fetch methods are the only ones supporting a conversion.

Sample fetch methods return data which can be of the following types :
  - boolean
  - integer (signed or unsigned)
  - IPv4 or IPv6 address
  - string
  - data block

Converters transform any of these data into any of these. For example, some converters might convert a string to a lower-case string while other ones would turn a string to an IPv4 address, or apply a netmask to an IP address. The resulting sample is of the type of the last converter applied to the list, which defaults to the type of the sample fetch method.

Each sample or converter returns data of a specific type, specified with its keyword in this documentation. When an ACL is declared using a standard sample fetch method, certain types automatically involved a default matching method which are summarized in the table below :

```
   +---------------------+-----------------+
   | Sample or converter | Default         |
   |    output type      | matching method |
   +---------------------+-----------------+
   | boolean             | bool            |
   +---------------------+-----------------+
   | integer             | int             |
   +---------------------+-----------------+
   | ip                  | ip              |
   +---------------------+-----------------+
   | string              | str             |
   +---------------------+-----------------+
   | binary              | none, use "-m"  |
   +---------------------+-----------------+
```

Note that in order to match a binary samples, it is mandatory to specify a matching method, see below.

The ACL engine can match these types against patterns of the following types :
  - boolean
  - integer or integer range
  - IP address / network
  - string (exact, substring, suffix, prefix, subdir, domain)
  - regular expression
  - hex block

The following ACL flags are currently supported :

```
   -i : ignore case during matching of all subsequent patterns.
   -f : load patterns from a file.
   -m : use a specific pattern matching method
   -n : forbid the DNS resolutions
   -M : load the file pointed by -f like a map file.
   -u : force the unique id of the ACL
   -- : force end of flags. Useful when a string looks like one of the flags.
```

The "-f" flag is followed by the name of a file from which all lines will be read as individual values. It is even possible to pass multiple "-f" arguments if the patterns are to be loaded from multiple files. Empty lines as well as

lines beginning with a sharp ('#') will be ignored. All leading spaces and tabs will be stripped. If it is absolutely necessary to insert a valid pattern beginning with a sharp, just prefix it with a space so that it is not taken for a comment. Depending on the data type and match method, haproxy may load the lines into a binary tree, allowing very fast lookups. This is true for IPv4 and exact string matching. In this case, duplicates will automatically be removed.

The "-M" flag allows an ACL to use a map file. If this flag is set, the file is parsed as two column file. The first column contains the patterns used by the ACL, and the second column contain the samples. The sample can be used later by a map. This can be useful in some rare cases where an ACL would just be used to check for the existence of a pattern in a map before a mapping is applied.

The "-u" flag forces the unique id of the ACL. This unique id is used with the socket interface to identify ACL and dynamically change its values. Note that a file is always identified by its name even if an id is set.

Also, note that the "-i" flag applies to subsequent entries and not to entries loaded from files preceding it. For instance :

```
    acl valid-ua hdr(user-agent) -f exact-ua.lst -i -f generic-ua.lst test
```

In this example, each line of "exact-ua.lst" will be exactly matched against the "user-agent" header of the request. Then each line of "generic-ua" will be case-insensitively matched. Then the word "test" will be insensitively matched as well.

The "-m" flag is used to select a specific pattern matching method on the input sample. All ACL-specific criteria imply a pattern matching method and generally do not need this flag. However, this flag is useful with generic sample fetch methods to describe how they're going to be matched against the patterns. This is required for sample fetches which return data type for which there is no obvious matching method (eg: string or binary). When "-m" is specified and followed by a pattern matching method name, this method is used instead of the default one for the criterion. This makes it possible to match contents in ways that were not initially planned, or with sample fetch methods which return a string. The matching method also affects the way the patterns are parsed.

The "-n" flag forbids the dns resolutions. It is used with the load of ip files. By default, if the parser cannot parse ip address it considers that the parsed string is maybe a domain name and try dns resolution. The flag "-n" disable this resolution. It is useful for detecting malformed ip lists. Note that if the DNS server is not reachable, the haproxy configuration parsing may last many minutes waiting fir the timeout. During this time no error messages are displayed. The flag "-n" disable this behavior. Note also that during the runtime, this function is disabled for the dynamic acl modifications.

There are some restrictions however. Not all methods can be used with all sample fetch methods. Also, if "-m" is used in conjunction with "-f", it must be placed first. The pattern matching method must be one of the following :

```
  - "found" : only check if the requested sample could be found in the stream,
              but do not compare it against any pattern. It is recommended not
              to pass any pattern to avoid confusion. This matching method is
              particularly useful to detect presence of certain contents such
              as headers, cookies, etc... even if they are empty and without
              comparing them to anything nor counting them.

  - "bool"  : check the value as a boolean. It can only be applied to fetches
              which return a boolean or integer value, and takes no pattern.
              Value zero or false does not match, all other values do match.

  - "int"   : match the value as an integer. It can be used with integer and
              boolean samples. Boolean false is integer 0, true is integer 1.
```

```
   - "ip"    : match the value as an IPv4 or IPv6 address. It is compatible
               with IP address samples only, so it is implied and never needed.

   - "bin"   : match the contents against an hexadecimal string representing a
               binary sequence. This may be used with binary or string samples.

   - "len"   : match the sample's length as an integer. This may be used with
               binary or string samples.

   - "str"   : exact match : match the contents against a string. This may be
               used with binary or string samples.

   - "sub"   : substring match : check that the contents contain at least one of
               the provided string patterns. This may be used with binary or
               string samples.

   - "reg"   : regex match : match the contents against a list of regular
               expressions. This may be used with binary or string samples.

   - "beg"   : prefix match : check that the contents begin like the provided
               string patterns. This may be used with binary or string samples.

   - "end"   : suffix match : check that the contents end like the provided
               string patterns. This may be used with binary or string samples.

   - "dir"   : subdir match : check that a slash-delimited portion of the
               contents exactly matches one of the provided string patterns.
               This may be used with binary or string samples.

   - "dom"   : domain match : check that a dot-delimited portion of the contents
               exactly match one of the provided string patterns. This may be
               used with binary or string samples.

For example, to quickly detect the presence of cookie "JSESSIONID" in an HTTP
request, it is possible to do :

    acl jsess_present cook(JSESSIONID) -m found

In order to apply a regular expression on the 500 first bytes of data in the
buffer, one would use the following acl :

    acl script_tag payload(0,500) -m reg -i <script>

On systems where the regex library is much slower when using "-i", it is
possible to convert the sample to lowercase before matching, like this :

    acl script_tag payload(0,500),lower -m reg <script>

All ACL-specific criteria imply a default matching method. Most often, these
criteria are composed by concatenating the name of the original sample fetch
method and the matching method. For example, "hdr_beg" applies the "beg" match
to samples retrieved using the "hdr" fetch method. Since all ACL-specific
criteria rely on a sample fetch method, it is always possible instead to use
the original sample fetch method and the explicit matching method using "-m".

If an alternate match is specified using "-m" on an ACL-specific criterion,
the matching method is simply applied to the underlying sample fetch method.
For example, all ACLs below are exact equivalent :

    acl short_form  hdr_beg(host)       www.
    acl alternate1  hdr_beg(host) -m beg www.
    acl alternate2  hdr_dom(host) -m beg www.
    acl alternate3  hdr(host)     -m beg www.
```


The table below summarizes the compatibility matrix between sample or converter types and the pattern types to fetch against. It indicates for each compatible combination the name of the matching method to be used, surrounded with angle brackets ">" and "<" when the method is the default one and will work by default without "-m".

	Input sample type				
pattern type	boolean	integer	ip	string	binary
none (presence only)	found	found	found	found	found
none (boolean value)	>bool<	bool		bool	
integer (value)	int	>int<	int	int	
integer (length)	len	len	len	len	len
IP address			>ip<	ip	ip
exact string	str	str	str	>str<	str
prefix	beg	beg	beg	beg	beg
suffix	end	end	end	end	end
substring	sub	sub	sub	sub	sub
subdir	dir	dir	dir	dir	dir
domain	dom	dom	dom	dom	dom
regex	reg	reg	reg	reg	reg
hex block				bin	bin

7.1.1. Matching booleans

In order to match a boolean, no value is needed and all values are ignored. Boolean matching is used by default for all fetch methods of type "boolean". When boolean matching is used, the fetched value is returned as-is, which means that a boolean "true" will always match and a boolean "false" will never match.

Boolean matching may also be enforced using "-m bool" on fetch methods which return an integer value. Then, integer value 0 is converted to the boolean "false" and all other values are converted to "true".

7.1.2. Matching integers

Integer matching applies by default to integer fetch methods. It can also be enforced on boolean fetches using "-m int". In this case, "false" is converted to the integer 0, and "true" is converted to the integer 1.

Integer matching also supports integer ranges and operators. Note that integer matching only applies to positive values. A range is a value expressed with a lower and an upper bound separated with a colon, both of which may be omitted.

For instance, "1024:65535" is a valid range to represent a range of unprivileged ports, and "1024:" would also work. "0:1023" is a valid

representation of privileged ports, and ":1023" would also work.

As a special case, some ACL functions support decimal numbers which are in fact two integers separated by a dot. This is used with some version checks for instance. All integer properties apply to those decimal numbers, including ranges and operators.

For an easier usage, comparison operators are also supported. Note that using operators with ranges does not make much sense and is strongly discouraged. Similarly, it does not make much sense to perform order comparisons with a set of values.

Available operators for integer matching are :

  eq : true if the tested value equals at least one value
  ge : true if the tested value is greater than or equal to at least one value
  gt : true if the tested value is greater than at least one value
  le : true if the tested value is less than or equal to at least one value
  lt : true if the tested value is less than at least one value

For instance, the following ACL matches any negative Content-Length header :

  acl negative-length hdr_val(content-length) lt 0

This one matches SSL versions between 3.0 and 3.1 (inclusive) :

  acl sslv3 req_ssl_ver 3:3.1

7.1.3. Matching strings

String matching applies to string or binary fetch methods, and exists in 6 different forms :

  - exact match   (-m str) : the extracted string must exactly match the
    patterns ;

  - substring match (-m sub) : the patterns are looked up inside the
    extracted string, and the ACL matches if any of them is found inside ;

  - prefix match   (-m beg) : the patterns are compared with the beginning of
    the extracted string, and the ACL matches if any of them matches.

  - suffix match   (-m end) : the patterns are compared with the end of the
    extracted string, and the ACL matches if any of them matches.

  - subdir match   (-m sub) : the patterns are looked up inside the extracted
    string, delimited with slashes ("/"), and the ACL matches if any of them
    matches.

  - domain match   (-m dom) : the patterns are looked up inside the extracted
    string, delimited with dots ("."), and the ACL matches if any of them
    matches.

String matching applies to verbatim strings as they are passed, with the exception of the backslash ("\") which makes it possible to escape some characters such as the space. If the "-i" flag is passed before the first string, then the matching will be performed ignoring the case. In order to match the string "-i", either set it second, or pass the "--" flag before the first string. Same applies of course to match the string "--".

7.1.4. Matching regular expressions (regexes)

Just like with string matching, regex matching applies to verbatim strings as they are passed, with the exception of the backslash ("\") which makes it possible to escape some characters such as the space. If the "-i" flag is passed before the first regex, then the matching will be performed ignoring the case. In order to match the string "-i", either set it second, or pass the "--" flag before the first string. Same principle applies of course to match the string "--".

7.1.5. Matching arbitrary data blocks

It is possible to match some extracted samples against a binary block which may not safely be represented as a string. For this, the patterns must be passed as a series of hexadecimal digits in an even number, when the match method is set to binary. Each sequence of two digits will represent a byte. The hexadecimal digits may be used upper or lower case.

Example :
    # match "Hello\n" in the input stream (\x48 \x65 \x6c \x6c \x6f \x0a)
    acl hello payload(0,6) -m bin 48656c6c6f0a

7.1.6. Matching IPv4 and IPv6 addresses

IPv4 addresses values can be specified either as plain addresses or with a netmask appended, in which case the IPv4 address matches whenever it is within the network. Plain addresses may also be replaced with a resolvable host name, but this practice is generally discouraged as it makes it more difficult to read and debug configurations. If hostnames are used, you should at least ensure that they are present in /etc/hosts so that the configuration does not depend on any random DNS match at the moment the configuration is parsed.

IPv6 may be entered in their usual form, with or without a netmask appended. Only bit counts are accepted for IPv6 netmasks. In order to avoid any risk of trouble with randomly resolved IP addresses, host names are never allowed in IPv6 patterns.

HAProxy is also able to match IPv4 addresses with IPv6 addresses in the following situations :
  - tested address is IPv4, pattern address is IPv4, the match applies
    in IPv4 using the supplied mask if any.
  - tested address is IPv6, pattern address is IPv6, the match applies
    in IPv6 using the supplied mask if any.
  - tested address is IPv6, pattern address is IPv4, the match applies in IPv4
    using the pattern's mask if the IPv6 address matches with 2002:IPV4::,
    ::IPV4 or ::ffff:IPV4, otherwise it fails.
  - tested address is IPv4, pattern address is IPv6, the IPv4 address is first
    converted to IPv6 by prefixing ::ffff: in front of it, then the match is
    applied in IPv6 using the supplied IPv6 mask.

7.2. Using ACLs to form conditions

Some actions are only performed upon a valid condition. A condition is a combination of ACLs with operators. 3 operators are supported :

  - AND (implicit)
  - OR (explicit with the "or" keyword or the "||" operator)
  - Negation with the exclamation mark ("!")

A condition is formed as a disjunctive form:

```
   [!]acl1 [!]acl2 ... [!]acln  { or [!]acl1 [!]acl2 ... [!]acln } ...
```

Such conditions are generally used after an "if" or "unless" statement,
indicating when the condition will trigger the action.

For instance, to block HTTP requests to the "*" URL with methods other than
"OPTIONS", as well as POST requests without content-length, and GET or HEAD
requests with a content-length greater than 0, and finally every request which
is not either GET/HEAD/POST/OPTIONS !

```
   acl missing_cl hdr_cnt(Content-length) eq 0
   block if HTTP_URL_STAR !METH_OPTIONS || METH_POST missing_cl
   block if METH_GET HTTP_CONTENT
   block unless METH_GET or METH_POST or METH_OPTIONS
```

To select a different backend for requests to static contents on the "www" site
and to every request on the "img", "video", "download" and "ftp" hosts :

```
   acl url_static  path_beg         /static /images /img /css
   acl url_static  path_end         .gif .png .jpg .css .js
   acl host_www    hdr_beg(host) -i www
   acl host_static hdr_beg(host) -i img. video. download. ftp.

   # now use backend "static" for all static-only hosts, and for static urls
   # of host "www". Use backend "www" for the rest.
   use_backend static if host_static or host_www url_static
   use_backend www    if host_www
```

It is also possible to form rules using "anonymous ACLs". Those are unnamed ACL
expressions that are built on the fly without needing to be declared. They must
be enclosed between braces, with a space before and after each brace (because
the braces must be seen as independent words). Example :

```
   The following rule :

       acl missing_cl hdr_cnt(Content-length) eq 0
       block if METH_POST missing_cl

   Can also be written that way :

       block if METH_POST { hdr_cnt(Content-length) eq 0 }
```

It is generally not recommended to use this construct because it's a lot easier
to leave errors in the configuration when written that way. However, for very
simple rules matching only one source IP address for instance, it can make more
sense to use them than to declare ACLs with random names. Another example of
good use is the following :

```
   With named ACLs :

        acl site_dead nbsrv(dynamic) lt 2
        acl site_dead nbsrv(static)  lt 2
        monitor fail  if site_dead

   With anonymous ACLs :

        monitor fail if { nbsrv(dynamic) lt 2 } || { nbsrv(static) lt 2 }
```

See section 4.2 for detailed help on the "block" and "use_backend" keywords.

7.3. Fetching samples

Historically, sample fetch methods were only used to retrieve data to match
against patterns using ACLs. With the arrival of stick-tables, a new class of
sample fetch methods was created, most often sharing the same syntax as their
ACL counterpart. These sample fetch methods are also known as "fetches". As
of now, ACLs and fetches have converged. All ACL fetch methods have been made
available as fetch methods, and ACLs may use any sample fetch method as well.

This section details all available sample fetch methods and their output type.
Some sample fetch methods have deprecated aliases that are used to maintain
compatibility with existing configurations. They are then explicitly marked as
deprecated and should not be used in new setups.

The ACL derivatives are also indicated when available, with their respective
matching methods. These ones all have a well defined default pattern matching
method, so it is never necessary (though allowed) to pass the "-m" option to
indicate how the sample will be matched using ACLs.

As indicated in the sample type versus matching compatibility matrix above,
when using a generic sample fetch method in an ACL, the "-m" option is
mandatory unless the sample type is one of boolean, integer, IPv4 or IPv6. When
the same keyword exists as an ACL keyword and as a standard fetch method, the
ACL engine will automatically pick the ACL-only one by default.

Some of these keywords support one or multiple mandatory arguments, and one or
multiple optional arguments. These arguments are strongly typed and are checked
when the configuration is parsed so that there is no risk of running with an
incorrect argument (eg: an unresolved backend name). Fetch function arguments
are passed between parenthesis and are delimited by commas. When an argument
is optional, it will be indicated below between square brackets ('[ ]'). When
all arguments are optional, the parenthesis may be omitted.

Thus, the syntax of a standard sample fetch method is one of the following :

```
   - name
   - name(arg1)
   - name(arg1,arg2)
```

7.3.1. Converters

Sample fetch methods may be combined with transformations to be applied on top
of the fetched sample (also called "converters"). These combinations form what
is called "sample expressions" and the result is a "sample". Initially this
was only supported by "stick on" and "stick store-request" directives but this
has now be extended to all places where samples may be used (acls, log-format,
unique-id-format, add-header, ...).

These transformations are enumerated as a series of specific keywords after the
sample fetch method. These keywords may equally be appended immediately after
the fetch keyword's argument, delimited by a comma. These keywords can also
support some arguments (eg: a netmask) which must be passed in parenthesis.

The currently available list of transformation keywords include :

base64
  Converts a binary input sample to a base64 string. It is used to log or
  transfer binary content in a way that can be reliably transferred (eg:
  an SSL ID can be copied in a header).

hex
  Converts a binary input sample to an hex string containing two hex digits per
  input byte. It is used to log or transfer hex dumps of some binary input data
  in a way that can be reliably transferred (eg: an SSL ID can be copied in a
  header).

```
10011
10012  http_date([<offset>])
10013    Converts an integer supposed to contain a date since epoch to a string
10014    representing this date in a format suitable for use in HTTP header fields. If
10015    an offset value is specified, then it is a number of seconds that is added to
10016    the date before the conversion is operated. This is particularly useful to
10017    emit Date header fields, Expires values in responses when combined with a
10018    positive offset, or Last-Modified values when the offset is negative.
10019
10020  ipmask(<mask>)
10021    Apply a mask to an IPv4 address, and use the result for lookups and storage.
10022    This can be used to make all hosts within a certain mask to share the same
10023    table entries and as such use the same server. The mask can be passed in
10024    dotted form (eg: 255.255.255.0) or in CIDR form (eg: 24).
10025
10026  language(<value>[,<default>])
10027    Returns the value with the highest q-factor from a list as extracted from the
10028    "accept-language" header using "req.fhdr". Values with no q-factor have a
10029    q-factor of 1. Values with a q-factor of 0 are dropped. Only values which
10030    belong to the list of semi-colon delimited <values> will be considered. The
10031    argument <value> syntax is "lang[;lang[;lang[;...]]]". If no value matches the
10032    given list and a default value is provided, it is returned. Note that language
10033    names may have a variant after a dash ('-'). If this variant is present in the
10034    list, it will be matched, but if it is not, only the base language is checked.
10035    The match is case-sensitive, and the output string is always one of those
10036    provided in arguments.  The ordering of arguments is meaningless, only the
10037    ordering of the values in the request counts, as the first value among
10038    multiple sharing the same q-factor is used.
10039
10040    Example :
10041
10042      # this configuration switches to the backend matching a
10043      # given language based on the request :
10044
10045      acl es req.fhdr(accept-language),language(es;fr;en) -m str es
10046      acl fr req.fhdr(accept-language),language(es;fr;en) -m str fr
10047      acl en req.fhdr(accept-language),language(es;fr;en) -m str en
10048      use_backend spanish if es
10049      use_backend french  if fr
10050      use_backend english if en
10051      default_backend choose_your_language
10052
10053  lower
10054    Convert a string sample to lower case. This can only be placed after a string
10055    sample fetch function or after a transformation keyword returning a string
10056    type. The result is of type string.
10057
10058  map(<map_file>[,<default_value>])
10059  map_<match_type>(<map_file>[,<default_value>])
10060  map_<match_type>_<output_type>(<map_file>[,<default_value>])
10061    Search the input value from <map_file> using the <match_type> matching method,
10062    and return the associated value converted to the type <output_type>. If the
10063    input value cannot be found in the <map_file>, the converter returns the
10064    <default_value>. If the <default_value> is not set, the converter fails and
10065    acts as if no input value could be fetched. If the <match_type> is not set, it
10066    defaults to "str". Likewise, if the <output_type> is not set, it defaults to
10067    "str". For convenience, the "map" keyword is an alias for "map_str" and maps a
10068    string to another string.
10069
10070    It is important to avoid overlapping between the keys : IP addresses and
10071    strings are stored in trees, so the first of the finest match will be used.
10072    Other keys are stored in lists, so the first matching occurrence will be used.
10073
10074    The following array contains the list of all map functions avalaible sorted by
10075    input type, match type and output type.
```



```
10076
10077    input type | match method | output type str | output type int | output type ip
10078    -----------+--------------+-----------------+-----------------+--------------
10079      str      | str          | map_str          | map_str_int     | map_str_ip
10080    -----------+--------------+-----------------+-----------------+--------------
10081      str      | beg          | map_beg          | map_beg_int     | map_end_ip
10082    -----------+--------------+-----------------+-----------------+--------------
10083      str      | sub          | map_sub          | map_sub_int     | map_sub_ip
10084    -----------+--------------+-----------------+-----------------+--------------
10085      str      | dir          | map_dir          | map_dir_int     | map_dir_ip
10086    -----------+--------------+-----------------+-----------------+--------------
10087      str      | dom          | map_dom          | map_dom_int     | map_dom_ip
10088    -----------+--------------+-----------------+-----------------+--------------
10089      str      | end          | map_end          | map_end_int     | map_end_ip
10090    -----------+--------------+-----------------+-----------------+--------------
10091      str      | reg          | map_reg          | map_reg_int     | map_reg_ip
10092    -----------+--------------+-----------------+-----------------+--------------
10093      int      | int          | map_int          | map_int_int     | map_int_ip
10094    -----------+--------------+-----------------+-----------------+--------------
10095      ip       | ip           | map_ip           | map_ip_int      | map_ip_ip
10096    -----------+--------------+-----------------+-----------------+--------------
10097
10098    The file contains one key + value per line. Lines which start with '#' are
10099    ignored, just like empty lines. Leading tabs and spaces are stripped. The key
10100    is then the first "word" (series of non-space/tabs characters), and the value
10101    is what follows this series of space/tab till the end of the line excluding
10102    trailing spaces/tabs.
10103
10104    Example :
10105
10106       # this is a comment and is ignored
10107          2.22.246.0/23    United Kingdom      \n
10108       <-><----------><--><------------><---->
10109        |       |       |         |        `- trailing spaces ignored
10110        |       |       |         `---------- value
10111        |       |       `-------------------- middle spaces ignored
10112        |       `---------------------------- key
10113        `------------------------------------ leading spaces ignored
10114
10115  upper
10116    Convert a string sample to upper case. This can only be placed after a string
10117    sample fetch function or after a transformation keyword returning a string
10118    type. The result is of type string.
10119
10120
10121  7.3.2. Fetching samples from internal states
10122  --------------------------------------------
10123
10124  A first set of sample fetch methods applies to internal information which does
10125  not even relate to any client information. These ones are sometimes used with
10126  "monitor-fail" directives to report an internal status to external watchers.
10127  The sample fetch methods described in this section are usable anywhere.
10128
10129  always_false : boolean
10130    Always returns the boolean "false" value. It may be used with ACLs as a
10131    temporary replacement for another one when adjusting configurations.
10132
10133  always_true : boolean
10134    Always returns the boolean "true" value. It may be used with ACLs as a
10135    temporary replacement for another one when adjusting configurations.
10136
10137  avg_queue([<backend>]) : integer
10138    Returns the total number of queued connections of the designated backend
10139    divided by the number of active servers. The current backend is used if no
10140    backend is specified. This is very similar to "queue" except that the size of
```

```
   the farm is considered, in order to give a more accurate measurement of the
   time it may take for a new connection to be processed. The main usage is with
   ACL to return a sorry page to new users when it becomes certain they will get
   a degraded service, or to pass to the backend servers in a header so that
   they decide to work in degraded mode or to disable some functions to speed up
   the processing a bit. Note that in the event there would not be any active
   server anymore, twice the number of queued connections would be considered as
   the measured value. This is a fair estimate, as we expect one server to get
   back soon anyway, but we still prefer to send new traffic to another backend
   if in better shape. See also the "queue", "be_conn", and "be_sess_rate"
   sample fetches.

be_conn([<backend>]) : integer
  Applies to the number of currently established connections on the backend,
  possibly including the connection being evaluated. If no backend name is
  specified, the current one is used. But it is also possible to check another
  backend. It can be used to use a specific farm when the nominal one is full.
  See also the "fe_conn", "queue" and "be_sess_rate" criteria.

be_sess_rate([<backend>]) : integer
  Returns an integer value corresponding to the sessions creation rate on the
  backend, in number of new sessions per second. This is used with ACLs to
  switch to an alternate backend when an expensive or fragile one reaches too
  high a session rate, or to limit abuse of service (eg. prevent sucking of an
  online dictionary). It can also be useful to add this element to logs using a
  log-format directive.

  Example :
        # Redirect to an error page if the dictionary is requested too often
        backend dynamic
            mode http
            acl being_scanned be_sess_rate gt 100
            redirect location /denied.html if being_scanned

connslots([<backend>]) : integer
  Returns an integer value corresponding to the number of connection slots
  still available in the backend, by totaling the maximum amount of
  connections on all servers and the maximum queue size. This is probably only
  used with ACLs.

  The basic idea here is to be able to measure the number of connection "slots"
  still available (connection + queue), so that anything beyond that (intended
  usage; see "use_backend" keyword) can be redirected to a different backend.

  'connslots' = number of available server connection slots, + number of
  available server queue slots.

  Note that while "fe_conn" may be used, "connslots" comes in especially
  useful when you have a case of traffic going to one single ip, splitting into
  multiple backends (perhaps using ACLs to do name-based load balancing) and
  you want to be able to differentiate between different backends, and their
  available "connslots".  Also, whereas "nbsrv" only measures servers that are
  actually *down*, this fetch is more fine-grained and looks into the number of
  available connection slots as well. See also "queue" and "avg_queue".

  OTHER CAVEATS AND NOTES: at this point in time, the code does not take care
  of dynamic connections. Also, if any of the server maxconn, or maxqueue is 0,
  then this fetch clearly does not make sense, in which case the value returned
  will be -1.

date([<offset>]) : integer
  Returns the current date as the epoch (number of seconds since 01/01/1970).
  If an offset value is specified, then it is a number of seconds that is added
  to the current date before returning the value. This is particularly useful
  to compute relative dates, as both positive and negative offsets are allowed.
```



```
  It is useful combined with the http_date converter.

  Example :

     # set an expires header to now+1 hour in every response
     http-response set-header Expires %[date(3600),http_date]

env(<name>) : string
  Returns a string containing the value of environment variable <name>. As a
  reminder, environment variables are per-process and are sampled when the
  process starts. This can be useful to pass some information to a next hop
  server, or with ACLs to take specific action when the process is started a
  certain way.

  Examples :
      # Pass the Via header to next hop with the local hostname in it
      http-request add-header Via 1.1\ %[env(HOSTNAME)]

      # reject cookie-less requests when the STOP environment variable is set
      http-request deny if !{ cook(SESSIONID) -m found } { env(STOP) -m found }

fe_conn([<frontend>]) : integer
  Returns the number of currently established connections on the frontend,
  possibly including the connection being evaluated. If no frontend name is
  specified, the current one is used. But it is also possible to check another
  frontend. It can be used to return a sorry page before hard-blocking, or to
  use a specific backend to drain new requests when the farm is considered
  full.  This is mostly used with ACLs but can also be used to pass some
  statistics to servers in HTTP headers. See also the "dst_conn", "be_conn",
  "fe_sess_rate" fetches.

fe_sess_rate([<frontend>]) : integer
  Returns an integer value corresponding to the sessions creation rate on the
  frontend, in number of new sessions per second. This is used with ACLs to
  limit the incoming session rate to an acceptable range in order to prevent
  abuse of service at the earliest moment, for example when combined with other
  layer 4 ACLs in order to force the clients to wait a bit for the rate to go
  down below the limit. It can also be useful to add this element to logs using
  a log-format directive. See also the "rate-limit sessions" directive for use
  in frontends.

  Example :
        # This frontend limits incoming mails to 10/s with a max of 100
        # concurrent connections. We accept any connection below 10/s, and
        # force excess clients to wait for 100 ms. Since clients are limited to
        # 100 max, there cannot be more than 10 incoming mails per second.
        frontend mail
            bind :25
            mode tcp
            maxconn 100
            acl too_fast fe_sess_rate ge 10
            tcp-request inspect-delay 100ms
            tcp-request content accept if ! too_fast
            tcp-request content accept if WAIT_END

nbproc : integer
  Returns an integer value corresponding to the number of processes that were
  started (it equals the global "nbproc" setting). This is useful for logging
  and debugging purposes.

nbsrv([<backend>]) : integer
  Returns an integer value corresponding to the number of usable servers of
  either the current backend or the named backend. This is mostly used with
  ACLs but can also be useful when added to logs. This is normally used to
  switch to an alternate backend when the number of servers is too low to
```

```
   to handle some load. It is useful to report a failure when combined with
   "monitor fail".

proc : integer
   Returns an integer value corresponding to the position of the process calling
   the function, between 1 and global.nbproc. This is useful for logging and
   debugging purposes.

queue([<backend>]) : integer
   Returns the total number of queued connections of the designated backend,
   including all the connections in server queues. If no backend name is
   specified, the current one is used, but it is also possible to check another
   one. This is useful with ACLs or to pass statistics to backend servers. This
   can be used to take actions when queuing goes above a known level, generally
   indicating a surge of traffic or a massive slowdown on the servers. One
   possible action could be to reject new users but still accept old ones. See
   also the "avg_queue", "be_conn", and "be_sess_rate" fetches.

rand([<range>]) : integer
   Returns a random integer value within a range of <range> possible values,
   starting at zero. If the range is not specified, it defaults to 2^32, which
   gives numbers between 0 and 4294967295. It can be useful to pass some values
   needed to take some routing decisions for example, or just for debugging
   purposes. This random must not be used for security purposes.

srv_conn([<backend>/]<server>) : integer
   Returns an integer value corresponding to the number of currently established
   connections on the designated server, possibly including the connection being
   evaluated. If <backend> is omitted, then the server is looked up in the
   current backend. It can be used to use a specific farm when one server is
   full, or to inform the server about our view of the number of active
   connections with it. See also the "fe_conn", "be_conn" and "queue" fetch
   methods.

srv_is_up([<backend>/]<server>) : boolean
   Returns true when the designated server is UP, and false when it is either
   DOWN or in maintenance mode. If <backend> is omitted, then the server is
   looked up in the current backend. It is mainly used to take action based on
   an external status reported via a health check (eg: a geographical site's
   availability). Another possible use which is more of a hack consists in
   using dummy servers as boolean variables that can be enabled or disabled from
   the CLI, so that rules depending on those ACLs can be tweaked in realtime.

srv_sess_rate([<backend>/]<server>) : integer
   Returns an integer corresponding to the sessions creation rate on the
   designated server, in number of new sessions per second. If <backend> is
   omitted, then the server is looked up in the current backend. This is mostly
   used with ACLs but can make sense with logs too. This is used to switch to an
   alternate backend when an expensive or fragile one reaches too high a session
   rate, or to limit abuse of service (eg. prevent latent requests from
   overloading servers).

   Example :
         # Redirect to a separate back
         acl srv1_full srv_sess_rate(be1/srv1) gt 50
         acl srv2_full srv_sess_rate(be1/srv2) gt 50
         use_backend be2 if srv1_full or srv2_full

stopping : boolean
   Returns TRUE if the process calling the function is currently stopping. This
   can be useful for logging, or for relaxing certain checks or helping close
   certain connections upon graceful shutdown.

table_avl([<table>]) : integer
   Returns the total number of available entries in the current proxy's
```



```
   stick-table or in the designated stick-table. See also table_cnt.

table_cnt([<table>]) : integer
   Returns the total number of entries currently in use in the current proxy's
   stick-table or in the designated stick-table. See also src_conn_cnt and
   table_avl for other entry counting methods.


7.3.3. Fetching samples at Layer 4
----------------------------------

The layer 4 usually describes just the transport layer which in haproxy is
closest to the connection, where no content is yet made available. The fetch
methods described here are usable as low as the "tcp-request connection" rule
sets unless they require some future information. Those generally include
TCP/IP addresses and ports, as well as elements from stick-tables related to
the incoming connection. For retrieving a value from a sticky counters, the
counter number can be explicitly set as 0, 1, or 2 using the pre-defined
"sc0_", "sc1_", or "sc2_" prefix, or it can be specified as the first integer
argument when using the "sc_" prefix. An optional table may be specified with
the "sc*" form, in which case the currently tracked key will be looked up into
this alternate table instead of the table currently being tracked.

be_id : integer
   Returns an integer containing the current backend's id. It can be used in
   frontends with responses to check which backend processed the request.

dst : ip
   This is the destination IPv4 address of the connection on the client side,
   which is the address the client connected to. It can be useful when running
   in transparent mode. It is of type IP and works on both IPv4 and IPv6 tables.
   On IPv6 tables, IPv4 address is mapped to its IPv6 equivalent, according to
   RFC 4291.

dst_conn : integer
   Returns an integer value corresponding to the number of currently established
   connections on the same socket including the one being evaluated. It is
   normally used with ACLs but can as well be used to pass the information to
   servers in an HTTP header or in logs. It can be used to either return a sorry
   page before hard-blocking, or to use a specific backend to drain new requests
   when the socket is considered saturated. This offers the ability to assign
   different limits to different listening ports or addresses. See also the
   "fe_conn" and "be_conn" fetches.

dst_port : integer
   Returns an integer value corresponding to the destination TCP port of the
   connection on the client side, which is the port the client connected to.
   This might be used when running in transparent mode, when assigning dynamic
   ports to some clients for a whole application session, to stick all users to
   a same server, or to pass the destination port information to a server using
   an HTTP header.

fe_id : integer
   Returns an integer containing the current frontend's id. It can be used in
   backends to check from which backend it was called, or to stick all users
   coming via a same frontend to the same server.

sc_bytes_in_rate(<ctr>[,<table>]) : integer
sc0_bytes_in_rate([<table>]) : integer
sc1_bytes_in_rate([<table>]) : integer
sc2_bytes_in_rate([<table>]) : integer
   Returns the average client-to-server bytes rate from the currently tracked
   counters, measured in amount of bytes over the period configured in the
   table. See also src_bytes_in_rate.
```

```
sc_bytes_out_rate(<ctr>[,<table>]) : integer
sc0_bytes_out_rate([<table>]) : integer
sc1_bytes_out_rate([<table>]) : integer
sc2_bytes_out_rate([<table>]) : integer
  Returns the average server-to-client bytes rate from the currently tracked
  counters, measured in amount of bytes over the period configured in the
  table. See also src_bytes_out_rate.

sc_clr_gpc0(<ctr>[,<table>]) : integer
sc0_clr_gpc0([<table>]) : integer
sc1_clr_gpc0([<table>]) : integer
sc2_clr_gpc0([<table>]) : integer
  Clears the first General Purpose Counter associated to the currently tracked
  counters, and returns its previous value. Before the first invocation, the
  stored value is zero, so first invocation will always return zero. This is
  typically used as a second ACL in an expression in order to mark a connection
  when a first ACL was verified :

        # block if 5 consecutive requests continue to come faster than 10 sess
        # per second, and reset the counter as soon as the traffic slows down.
        acl abuse sc0_http_req_rate gt 10
        acl kill  sc0_inc_gpc0 gt 5
        acl save  sc0_clr_gpc0 ge 0
        tcp-request connection accept if !abuse save
        tcp-request connection reject if abuse kill

sc_conn_cnt(<ctr>[,<table>]) : integer
sc0_conn_cnt([<table>]) : integer
sc1_conn_cnt([<table>]) : integer
sc2_conn_cnt([<table>]) : integer
  Returns the cumulated number of incoming connections from currently tracked
  counters. See also src_conn_cnt.

sc_conn_cur(<ctr>[,<table>]) : integer
sc0_conn_cur([<table>]) : integer
sc1_conn_cur([<table>]) : integer
sc2_conn_cur([<table>]) : integer
  Returns the current amount of concurrent connections tracking the same
  tracked counters. This number is automatically incremented when tracking
  begins and decremented when tracking stops. See also src_conn_cur.

sc_conn_rate(<ctr>[,<table>]) : integer
sc0_conn_rate([<table>]) : integer
sc1_conn_rate([<table>]) : integer
sc2_conn_rate([<table>]) : integer
  Returns the average connection rate from the currently tracked counters,
  measured in amount of connections over the period configured in the table.
  See also src_conn_rate.

sc_get_gpc0(<ctr>[,<table>]) : integer
sc0_get_gpc0([<table>]) : integer
sc1_get_gpc0([<table>]) : integer
sc2_get_gpc0([<table>]) : integer
  Returns the value of the first General Purpose Counter associated to the
  currently tracked counters. See also src_get_gpc0 and sc/sc0/sc1/sc2_inc_gpc0.

sc_gpc0_rate(<ctr>[,<table>]) : integer
sc0_gpc0_rate([<table>]) : integer
sc1_gpc0_rate([<table>]) : integer
sc2_gpc0_rate([<table>]) : integer
  Returns the average increment rate of the first General Purpose Counter
  associated to the currently tracked counters. It reports the frequency
  which the gpc0 counter was incremented over the configured period. See also
  src_gpc0_rate, sc/sc0/sc1/sc2_get_gpc0, and sc/sc0/sc1/sc2_inc_gpc0. Note
  that the "gpc0_rate" counter must be stored in the stick-table for a value to
```



```
  be returned, as "gpc0" only holds the event count.

sc_http_err_cnt(<ctr>[,<table>]) : integer
sc0_http_err_cnt([<table>]) : integer
sc1_http_err_cnt([<table>]) : integer
sc2_http_err_cnt([<table>]) : integer
  Returns the cumulated number of HTTP errors from the currently tracked
  counters. This includes the both request errors and 4xx error responses.
  See also src_http_err_cnt.

sc_http_err_rate(<ctr>[,<table>]) : integer
sc0_http_err_rate([<table>]) : integer
sc1_http_err_rate([<table>]) : integer
sc2_http_err_rate([<table>]) : integer
  Returns the average rate of HTTP errors from the currently tracked counters,
  measured in amount of errors over the period configured in the table. This
  includes the both request errors and 4xx error responses. See also
  src_http_err_rate.

sc_http_req_cnt(<ctr>[,<table>]) : integer
sc0_http_req_cnt([<table>]) : integer
sc1_http_req_cnt([<table>]) : integer
sc2_http_req_cnt([<table>]) : integer
  Returns the cumulated number of HTTP requests from the currently tracked
  counters. This includes every started request, valid or not. See also
  src_http_req_cnt.

sc_http_req_rate(<ctr>[,<table>]) : integer
sc0_http_req_rate([<table>]) : integer
sc1_http_req_rate([<table>]) : integer
sc2_http_req_rate([<table>]) : integer
  Returns the average rate of HTTP requests from the currently tracked
  counters, measured in amount of requests over the period configured in
  the table. This includes every started request, valid or not. See also
  src_http_req_rate.

sc_inc_gpc0(<ctr>[,<table>]) : integer
sc0_inc_gpc0([<table>]) : integer
sc1_inc_gpc0([<table>]) : integer
sc2_inc_gpc0([<table>]) : integer
  Increments the first General Purpose Counter associated to the currently
  tracked counters, and returns its new value. Before the first invocation,
  the stored value is zero, so first invocation will increase it to 1 and will
  return 1. This is typically used as a second ACL in an expression in order
  to mark a connection when a first ACL was verified :

        acl abuse sc0_http_req_rate gt 10
        acl kill  sc0_inc_gpc0 gt 0
        tcp-request connection reject if abuse kill

sc_kbytes_in(<ctr>[,<table>]) : integer
sc0_kbytes_in([<table>]) : integer
sc1_kbytes_in([<table>]) : integer
sc2_kbytes_in([<table>]) : integer
  Returns the total amount of client-to-server data from the currently tracked
  counters, measured in kilobytes. The test is currently performed on 32-bit
  integers, which limits values to 4 terabytes. See also src_kbytes_in.

sc_kbytes_out(<ctr>[,<table>]) : integer
sc0_kbytes_out([<table>]) : integer
sc1_kbytes_out([<table>]) : integer
sc2_kbytes_out([<table>]) : integer
  Returns the total amount of server-to-client data from the currently tracked
  counters, measured in kilobytes. The test is currently performed on 32-bit
  integers, which limits values to 4 terabytes. See also src_kbytes_out.
```

```
10531
10532  sc_sess_cnt(<ctr>[,<table>]) : integer
10533  sc0_sess_cnt([<table>]) : integer
10534  sc1_sess_cnt([<table>]) : integer
10535  sc2_sess_cnt([<table>]) : integer
10536    Returns the cumulated number of incoming connections that were transformed
10537    into sessions, which means that they were accepted by a "tcp-request
10538    connection" rule, from the currently tracked counters. A backend may count
10539    more sessions than connections because each connection could result in many
10540    backend sessions if some HTTP keep-alive is performed over the connection
10541    with the client. See also src_sess_cnt.
10542
10543  sc_sess_rate(<ctr>[,<table>]) : integer
10544  sc0_sess_rate([<table>]) : integer
10545  sc1_sess_rate([<table>]) : integer
10546  sc2_sess_rate([<table>]) : integer
10547    Returns the average session rate from the currently tracked counters,
10548    measured in amount of sessions over the period configured in the table. A
10549    session is a connection that got past the early "tcp-request connection"
10550    rules. A backend may count more sessions than connections because each
10551    connection could result in many backend sessions if some HTTP keep-alive is
10552    performed over the connection with the client. See also src_sess_rate.
10553
10554  sc_tracked(<ctr>[,<table>]) : boolean
10555  sc0_tracked([<table>]) : boolean
10556  sc1_tracked([<table>]) : boolean
10557  sc2_tracked([<table>]) : boolean
10558    Returns true if the designated session counter is currently being tracked by
10559    the current session. This can be useful when deciding whether or not we want
10560    to set some values in a header passed to the server.
10561
10562  sc_trackers(<ctr>[,<table>]) : integer
10563  sc0_trackers([<table>]) : integer
10564  sc1_trackers([<table>]) : integer
10565  sc2_trackers([<table>]) : integer
10566    Returns the current amount of concurrent connections tracking the same
10567    tracked counters. This number is automatically incremented when tracking
10568    begins and decremented when tracking stops. It differs from sc0_conn_cur in
10569    that it does not rely on any stored information but on the table's reference
10570    count (the "use" value which is returned by "show table" on the CLI). This
10571    may sometimes be more suited for layer7 tracking. It can be used to tell a
10572    server how many concurrent connections there are from a given address for
10573    example.
10574
10575  so_id : integer
10576    Returns an integer containing the current listening socket's id. It is useful
10577    in frontends involving many "bind" lines, or to stick all users coming via a
10578    same socket to the same server.
10579
10580  src : ip
10581    This is the source IPv4 address of the client of the session.  It is of type
10582    IP and works on both IPv4 and IPv6 tables. On IPv6 tables, IPv4 addresses are
10583    mapped to their IPv6 equivalent, according to RFC 4291. Note that it is the
10584    TCP-level source address which is used, and not the address of a client
10585    behind a proxy. However if the "accept-proxy" bind directive is used, it can
10586    be the address of a client behind another PROXY-protocol compatible component
10587    for all rule sets except "tcp-request connection" which sees the real address.
10588
10589    Example:
10590         # add an HTTP header in requests with the originating address' country
10591         http-request set-header X-Country %[src,map_ip(geoip.lst)]
10592
10593  src_bytes_in_rate([<table>]) : integer
10594    Returns the average bytes rate from the incoming connection's source address
10595    in the current proxy's stick-table or in the designated stick-table, measured
```



```
10596    in amount of bytes over the period configured in the table. If the address is
10597    not found, zero is returned. See also sc/sc0/sc1/sc2_bytes_in_rate.
10598
10599  src_bytes_out_rate([<table>]) : integer
10600    Returns the average bytes rate to the incoming connection's source address in
10601    the current proxy's stick-table or in the designated stick-table, measured in
10602    amount of bytes over the period configured in the table. If the address is
10603    not found, zero is returned. See also sc/sc0/sc1/sc2_bytes_out_rate.
10604
10605  src_clr_gpc0([<table>]) : integer
10606    Clears the first General Purpose Counter associated to the incoming
10607    connection's source address in the current proxy's stick-table or in the
10608    designated stick-table, and returns its previous value. If the address is not
10609    found, an entry is created and 0 is returned. This is typically used as a
10610    second ACL in an expression in order to mark a connection when a first ACL
10611    was verified :
10612
10613         # block if 5 consecutive requests continue to come faster than 10 sess
10614         # per second, and reset the counter as soon as the traffic slows down.
10615         acl abuse src_http_req_rate gt 10
10616         acl kill  src_inc_gpc0 gt 5
10617         acl save  src_clr_gpc0 ge 0
10618         tcp-request connection accept if !abuse save
10619         tcp-request connection reject if abuse kill
10620
10621  src_conn_cnt([<table>]) : integer
10622    Returns the cumulated number of connections initiated from the current
10623    incoming connection's source address in the current proxy's stick-table or in
10624    the designated stick-table. If the address is not found, zero is returned.
10625    See also sc/sc0/sc1/sc2_conn_cnt.
10626
10627  src_conn_cur([<table>]) : integer
10628    Returns the current amount of concurrent connections initiated from the
10629    current incoming connection's source address in the current proxy's
10630    stick-table or in the designated stick-table. If the address is not found,
10631    zero is returned. See also sc/sc0/sc1/sc2_conn_cur.
10632
10633  src_conn_rate([<table>]) : integer
10634    Returns the average connection rate from the incoming connection's source
10635    address in the current proxy's stick-table or in the designated stick-table,
10636    measured in amount of connections over the period configured in the table. If
10637    the address is not found, zero is returned. See also sc/sc0/sc1/sc2_conn_rate.
10638
10639  src_get_gpc0([<table>]) : integer
10640    Returns the value of the first General Purpose Counter associated to the
10641    incoming connection's source address in the current proxy's stick-table or in
10642    the designated stick-table. If the address is not found, zero is returned.
10643    See also sc/sc0/sc1/sc2_get_gpc0 and src_inc_gpc0.
10644
10645  src_gpc0_rate([<table>]) : integer
10646    Returns the average increment rate of the first General Purpose Counter
10647    associated to the incoming connection's source address in the current proxy's
10648    stick-table or in the designated stick-table. It reports the frequency
10649    which the gpc0 counter was incremented over the configured period. See also
10650    sc/sc0/sc1/sc2_gpc0_rate, src_get_gpc0, and sc/sc0/sc1/sc2_inc_gpc0. Note
10651    that the "gpc0_rate" counter must be stored in the stick-table for a value to
10652    be returned, as "gpc0" only holds the event count.
10653
10654  src_http_err_cnt([<table>]) : integer
10655    Returns the cumulated number of HTTP errors from the incoming connection's
10656    source address in the current proxy's stick-table or in the designated
10657    stick-table. This includes the both request errors and 4xx error responses.
10658    See also sc/sc0/sc1/sc2_http_err_cnt. If the address is not found, zero is
10659    returned.
10660
```

```
10661  src_http_err_rate([<table>]) : integer
10662    Returns the average rate of HTTP errors from the incoming connection's source
10663    address in the current proxy's stick-table or in the designated stick-table,
10664    measured in amount of errors over the period configured in the table. This
10665    includes the both request errors and 4xx error responses. If the address is
10666    not found, zero is returned. See also sc/sc0/sc1/sc2_http_err_rate.
10667
10668  src_http_req_cnt([<table>]) : integer
10669    Returns the cumulated number of HTTP requests from the incoming connection's
10670    source address in the current proxy's stick-table or in the designated stick-
10671    table. This includes every started request, valid or not. If the address is
10672    not found, zero is returned. See also sc/sc0/sc1/sc2_http_req_cnt.
10673
10674  src_http_req_rate([<table>]) : integer
10675    Returns the average rate of HTTP requests from the incoming connection's
10676    source address in the current proxy's stick-table or in the designated stick-
10677    table, measured in amount of requests over the period configured in the
10678    table. This includes every started request, valid or not. If the address is
10679    not found, zero is returned. See also sc/sc0/sc1/sc2_http_req_rate.
10680
10681  src_inc_gpc0([<table>]) : integer
10682    Increments the first General Purpose Counter associated to the incoming
10683    connection's source address in the current proxy's stick-table or in the
10684    designated stick-table, and returns its new value. If the address is not
10685    found, an entry is created and 1 is returned. See also sc0/sc2/sc2_inc_gpc0.
10686    This is typically used as a second ACL in an expression in order to mark a
10687    connection when a first ACL was verified :
10688
10689          acl abuse src_http_req_rate gt 10
10690          acl kill  src_inc_gpc0 gt 0
10691          tcp-request connection reject if abuse kill
10692
10693  src_kbytes_in([<table>]) : integer
10694    Returns the total amount of data received from the incoming connection's
10695    source address in the current proxy's stick-table or in the designated
10696    stick-table, measured in kilobytes. If the address is not found, zero is
10697    returned. The test is currently performed on 32-bit integers, which limits
10698    values to 4 terabytes. See also sc/sc0/sc1/sc2_kbytes_in.
10699
10700  src_kbytes_out([<table>]) : integer
10701    Returns the total amount of data sent to the incoming connection's source
10702    address in the current proxy's stick-table or in the designated stick-table,
10703    measured in kilobytes. If the address is not found, zero is returned. The
10704    test is currently performed on 32-bit integers, which limits values to 4
10705    terabytes. See also sc/sc0/sc1/sc2_kbytes_out.
10706
10707  src_port : integer
10708    Returns an integer value corresponding to the TCP source port of the
10709    connection on the client side, which is the port the client connected from.
10710    Usage of this function is very limited as modern protocols do not care much
10711    about source ports nowadays.
10712
10713  src_sess_cnt([<table>]) : integer
10714    Returns the cumulated number of connections initiated from the incoming
10715    connection's source IPv4 address in the current proxy's stick-table or in the
10716    designated stick-table, that were transformed into sessions, which means that
10717    they were accepted by "tcp-request" rules. If the address is not found, zero
10718    is returned. See also sc/sc0/sc1/sc2_sess_cnt.
10719
10720  src_sess_rate([<table>]) : integer
10721    Returns the average session rate from the incoming connection's source
10722    address in the current proxy's stick-table or in the designated stick-table,
10723    measured in amount of sessions over the period configured in the table. A
10724    session is a connection that went past the early "tcp-request" rules. If the
10725    address is not found, zero is returned. See also sc/sc0/sc1/sc2_sess_rate.
```



```
10726
10727  src_updt_conn_cnt([<table>]) : integer
10728    Creates or updates the entry associated to the incoming connection's source
10729    address in the current proxy's stick-table or in the designated stick-table.
10730    This table must be configured to store the "conn_cnt" data type, otherwise
10731    the match will be ignored. The current count is incremented by one, and the
10732    expiration timer refreshed. The updated count is returned, so this match
10733    can't return zero. This was used to reject service abusers based on their
10734    source address. Note: it is recommended to use the more complete "track-sc*"
10735    actions in "tcp-request" rules instead.
10736
10737    Example :
10738          # This frontend limits incoming SSH connections to 3 per 10 second for
10739          # each source address, and rejects excess connections until a 10 second
10740          # silence is observed. At most 20 addresses are tracked.
10741          listen ssh
10742              bind :22
10743              mode tcp
10744              maxconn 100
10745              stick-table type ip size 20 expire 10s store conn_cnt
10746              tcp-request content reject if { src_updt_conn_cnt gt 3 }
10747              server local 127.0.0.1:22
10748
10749  srv_id : integer
10750    Returns an integer containing the server's id when processing the response.
10751    While it's almost only used with ACLs, it may be used for logging or
10752    debugging.
10753
10754
10755  7.3.4. Fetching samples at Layer 5
10756  ----------------------------------
10757
10758  The layer 5 usually describes just the session layer which in haproxy is
10759  closest to the session once all the connection handshakes are finished, but
10760  when no content is yet made available. The fetch methods described here are
10761  usable as low as the "tcp-request content" rule sets unless they require some
10762  future information. Those generally include the results of SSL negotiations.
10763
10764  ssl_bc : boolean
10765    Returns true when the back connection was made via an SSL/TLS transport
10766    layer and is locally deciphered. This means the outgoing connection was made
10767    other a server with the "ssl" option.
10768
10769  ssl_bc_alg_keysize : integer
10770    Returns the symmetric cipher key size supported in bits when the outgoing
10771    connection was made over an SSL/TLS transport layer.
10772
10773  ssl_bc_cipher : string
10774    Returns the name of the used cipher when the outgoing connection was made
10775    over an SSL/TLS transport layer.
10776
10777  ssl_bc_protocol : string
10778    Returns the name of the used protocol when the outgoing connection was made
10779    over an SSL/TLS transport layer.
10780
10781  ssl_bc_unique_id : binary
10782    When the outgoing connection was made over an SSL/TLS transport layer,
10783    returns the TLS unique ID as defined in RFC5929 section 3. The unique id
10784    can be encoded to base64 using the converter: "ssl_bc_unique_id,base64".
10785
10786  ssl_bc_session_id : binary
10787    Returns the SSL ID of the back connection when the outgoing connection was
10788    made over an SSL/TLS transport layer. It is useful to log if we want to know
10789    if session was reused or not.
10790
```

```
ssl_bc_use_keysize : integer
  Returns the symmetric cipher key size used in bits when the outgoing
  connection was made over an SSL/TLS transport layer.

ssl_c_ca_err : integer
  When the incoming connection was made over an SSL/TLS transport layer,
  returns the ID of the first error detected during verification of the client
  certificate at depth > 0, or 0 if no error was encountered during this
  verification process. Please refer to your SSL library's documentation to
  find the exhaustive list of error codes.

ssl_c_ca_err_depth : integer
  When the incoming connection was made over an SSL/TLS transport layer,
  returns the depth in the CA chain of the first error detected during the
  verification of the client certificate. If no error is encountered, 0 is
  returned.

ssl_c_der : binary
  Returns the DER formatted certificate presented by the client when the
  incoming connection was made over an SSL/TLS transport layer. When used for
  an ACL, the value(s) to match against can be passed in hexadecimal form.

ssl_c_err : integer
  When the incoming connection was made over an SSL/TLS transport layer,
  returns the ID of the first error detected during verification at depth 0, or
  0 if no error was encountered during this verification process. Please refer
  to your SSL library's documentation to find the exhaustive list of error
  codes.

ssl_c_i_dn([<entry>[,<occ>]]) : string
  When the incoming connection was made over an SSL/TLS transport layer,
  returns the full distinguished name of the issuer of the certificate
  presented by the client when no <entry> is specified, or the value of the
  first given entry found from the beginning of the DN. If a positive/negative
  occurrence number is specified as the optional second argument, it returns
  the value of the nth given entry value from the beginning/end of the DN.
  For instance, "ssl_c_i_dn(OU,2)" the second organization unit, and
  "ssl_c_i_dn(CN)" retrieves the common name.

ssl_c_key_alg : string
  Returns the name of the algorithm used to generate the key of the certificate
  presented by the client when the incoming connection was made over an SSL/TLS
  transport layer.

ssl_c_notafter : string
  Returns the end date presented by the client as a formatted string
  YYMMDDhhmmss[Z] when the incoming connection was made over an SSL/TLS
  transport layer.

ssl_c_notbefore : string
  Returns the start date presented by the client as a formatted string
  YYMMDDhhmmss[Z] when the incoming connection was made over an SSL/TLS
  transport layer.

ssl_c_s_dn([<entry>[,<occ>]]) : string
  When the incoming connection was made over an SSL/TLS transport layer,
  returns the full distinguished name of the subject of the certificate
  presented by the client when no <entry> is specified, or the value of the
  first given entry found from the beginning of the DN. If a positive/negative
  occurrence number is specified as the optional second argument, it returns
  the value of the nth given entry value from the beginning/end of the DN.
  For instance, "ssl_c_s_dn(OU,2)" the second organization unit, and
  "ssl_c_s_dn(CN)" retrieves the common name.

ssl_c_serial : binary
```



```
  Returns the serial of the certificate presented by the client when the
  incoming connection was made over an SSL/TLS transport layer. When used for
  an ACL, the value(s) to match against can be passed in hexadecimal form.

ssl_c_sha1 : binary
  Returns the SHA-1 fingerprint of the certificate presented by the client when
  the incoming connection was made over an SSL/TLS transport layer. This can be
  used to stick a client to a server, or to pass this information to a server.
  Note that the output is binary, so if you want to pass that signature to the
  server, you need to encode it in hex or base64, such as in the example below:

     http-request set-header X-SSL-Client-SHA1 %[ssl_c_sha1,hex]

ssl_c_sig_alg : string
  Returns the name of the algorithm used to sign the certificate presented by
  the client when the incoming connection was made over an SSL/TLS transport
  layer.

ssl_c_used : boolean
  Returns true if current SSL session uses a client certificate even if current
  connection uses SSL session resumption. See also "ssl_fc_has_crt".

ssl_c_verify : integer
  Returns the verify result error ID when the incoming connection was made over
  an SSL/TLS transport layer, otherwise zero if no error is encountered. Please
  refer to your SSL library's documentation for an exhaustive list of error
  codes.

ssl_c_version : integer
  Returns the version of the certificate presented by the client when the
  incoming connection was made over an SSL/TLS transport layer.

ssl_f_der : binary
  Returns the DER formatted certificate presented by the frontend when the
  incoming connection was made over an SSL/TLS transport layer. When used for
  an ACL, the value(s) to match against can be passed in hexadecimal form.

ssl_f_i_dn([<entry>[,<occ>]]) : string
  When the incoming connection was made over an SSL/TLS transport layer,
  returns the full distinguished name of the issuer of the certificate
  presented by the frontend when no <entry> is specified, or the value of the
  first given entry found from the beginning of the DN. If a positive/negative
  occurrence number is specified as the optional second argument, it returns
  the value of the nth given entry value from the beginning/end of the DN.
  For instance, "ssl_f_i_dn(OU,2)" the second organization unit, and
  "ssl_f_i_dn(CN)" retrieves the common name.

ssl_f_key_alg : string
  Returns the name of the algorithm used to generate the key of the certificate
  presented by the frontend when the incoming connection was made over an
  SSL/TLS transport layer.

ssl_f_notafter : string
  Returns the end date presented by the frontend as a formatted string
  YYMMDDhhmmss[Z] when the incoming connection was made over an SSL/TLS
  transport layer.

ssl_f_notbefore : string
  Returns the start date presented by the frontend as a formatted string
  YYMMDDhhmmss[Z] when the incoming connection was made over an SSL/TLS
  transport layer.

ssl_f_s_dn([<entry>[,<occ>]]) : string
  When the incoming connection was made over an SSL/TLS transport layer,
  returns the full distinguished name of the subject of the certificate
```

```
10921    presented by the frontend when no <entry> is specified, or the value of the
10922    first given entry found from the beginning of the DN. If a positive/negative
10923    occurrence number is specified as the optional second argument, it returns
10924    the value of the nth given entry value from the beginning/end of the DN.
10925    For instance, "ssl_f_s_dn(OU,2)" the second organization unit, and
10926    "ssl_f_s_dn(CN)" retrieves the common name.
10927
10928  ssl_f_serial : binary
10929    Returns the serial of the certificate presented by the frontend when the
10930    incoming connection was made over an SSL/TLS transport layer. When used for
10931    an ACL, the value(s) to match against can be passed in hexadecimal form.
10932
10933  ssl_f_sha1 : binary
10934    Returns the SHA-1 fingerprint of the certificate presented by the frontend
10935    when the incoming connection was made over an SSL/TLS transport layer. This
10936    can be used to know which certificate was chosen using SNI.
10937
10938  ssl_f_sig_alg : string
10939    Returns the name of the algorithm used to sign the certificate presented by
10940    the frontend when the incoming connection was made over an SSL/TLS transport
10941    layer.
10942
10943  ssl_f_version : integer
10944    Returns the version of the certificate presented by the frontend when the
10945    incoming connection was made over an SSL/TLS transport layer.
10946
10947  ssl_fc : boolean
10948    Returns true when the front connection was made via an SSL/TLS transport
10949    layer and is locally deciphered. This means it has matched a socket declared
10950    with a "bind" line having the "ssl" option.
10951
10952    Example :
10953          # This passes "X-Proto: https" to servers when client connects over SSL
10954          listen http-https
10955              bind :80
10956              bind :443 ssl crt /etc/haproxy.pem
10957              http-request add-header X-Proto https if { ssl_fc }
10958
10959  ssl_fc_alg_keysize : integer
10960    Returns the symmetric cipher key size supported in bits when the incoming
10961    connection was made over an SSL/TLS transport layer.
10962
10963  ssl_fc_alpn : string
10964    This extracts the Application Layer Protocol Negotiation field from an
10965    incoming connection made via a TLS transport layer and locally deciphered by
10966    haproxy. The result is a string containing the protocol name advertised by
10967    the client. The SSL library must have been built with support for TLS
10968    extensions enabled (check haproxy -vv). Note that the TLS ALPN extension is
10969    not advertised unless the "alpn" keyword on the "bind" line specifies a
10970    protocol list. Also, nothing forces the client to pick a protocol from this
10971    list, any other one may be requested. The TLS ALPN extension is meant to
10972    replace the TLS NPN extension. See also "ssl_fc_npn".
10973
10974  ssl_fc_cipher : string
10975    Returns the name of the used cipher when the incoming connection was made
10976    over an SSL/TLS transport layer.
10977
10978  ssl_fc_has_crt : boolean
10979    Returns true if a client certificate is present in an incoming connection over
10980    SSL/TLS transport layer. Useful if 'verify' statement is set to 'optional'.
10981    Note: on SSL session resumption with Session ID or TLS ticket, client
10982    certificate is not present in the current connection but may be retrieved
10983    from the cache or the ticket. So prefer "ssl_c_used" if you want to check if
10984    current SSL session uses a client certificate.
10985
```



```
10986  ssl_fc_has_sni : boolean
10987    This checks for the presence of a Server Name Indication TLS extension (SNI)
10988    in an incoming connection was made over an SSL/TLS transport layer. Returns
10989    true when the incoming connection presents a TLS SNI field. This requires
10990    that the SSL library is build with support for TLS extensions enabled (check
10991    haproxy -vv).
10992
10993  ssl_fc_npn : string
10994    This extracts the Next Protocol Negotiation field from an incoming connection
10995    made via a TLS transport layer and locally deciphered by haproxy. The result
10996    is a string containing the protocol name advertised by the client. The SSL
10997    library must have been built with support for TLS extensions enabled (check
10998    haproxy -vv). Note that the TLS NPN extension is not advertised unless the
10999    "npn" keyword on the "bind" line specifies a protocol list. Also, nothing
11000    forces the client to pick a protocol from this list, any other one may be
11001    requested. Please note that the TLS NPN extension was replaced with ALPN.
11002
11003  ssl_fc_protocol : string
11004    Returns the name of the used protocol when the incoming connection was made
11005    over an SSL/TLS transport layer.
11006
11007  ssl_fc_unique_id : binary
11008    When the incoming connection was made over an SSL/TLS transport layer,
11009    returns the TLS unique ID as defined in RFC5929 section 3. The unique id
11010    can be encoded to base64 using the converter: "ssl_bc_unique_id,base64".
11011
11012  ssl_fc_session_id : binary
11013    Returns the SSL ID of the front connection when the incoming connection was
11014    made over an SSL/TLS transport layer. It is useful to stick a given client to
11015    a server. It is important to note that some browsers refresh their session ID
11016    every few minutes.
11017
11018  ssl_fc_sni : string
11019    This extracts the Server Name Indication TLS extension (SNI) field from an
11020    incoming connection made via an SSL/TLS transport layer and locally
11021    deciphered by haproxy. The result (when present) typically is a string
11022    matching the HTTPS host name (253 chars or less). The SSL library must have
11023    been built with support for TLS extensions enabled (check haproxy -vv).
11024
11025    This fetch is different from "req_ssl_sni" above in that it applies to the
11026    connection being deciphered by haproxy and not to SSL contents being blindly
11027    forwarded. See also "ssl_fc_sni_end" and "ssl_fc_sni_reg" below. This
11028    requires that the SSL library is build with support for TLS extensions
11029    enabled (check haproxy -vv).
11030
11031    ACL derivatives :
11032      ssl_fc_sni_end : suffix match
11033      ssl_fc_sni_reg : regex match
11034
11035  ssl_fc_use_keysize : integer
11036    Returns the symmetric cipher key size used in bits when the incoming
11037    connection was made over an SSL/TLS transport layer.
11038
11039
11040  7.3.5. Fetching samples from buffer contents (Layer 6)
11041  ------------------------------------------------------
11042
11043  Fetching samples from buffer contents is a bit different from the previous
11044  sample fetches above because the sampled data are ephemeral. These data can
11045  only be used when they're available and will be lost when they're forwarded.
11046  For this reason, samples fetched from buffer contents during a request cannot
11047  be used in a response for example. Even while the data are being fetched, they
11048  can change. Sometimes it is necessary to set some delays or combine multiple
11049  sample fetch methods to ensure that the expected data are complete and usable,
11050  for example through TCP request content inspection. Please see the "tcp-request
```

```
content" keyword for more detailed information on the subject.

payload(<offset>,<length>) : binary (deprecated)
  This is an alias for "req.payload" when used in the context of a request (eg:
  "stick on", "stick match"), and for "res.payload" when used in the context of
  a response such as in "stick store response".

payload_lv(<offset1>,<length>[,<offset2>]) : binary (deprecated)
  This is an alias for "req.payload_lv" when used in the context of a request
  (eg: "stick on", "stick match"), and for "res.payload_lv" when used in the
  context of a response such as in "stick store response".

req.len : integer
req_len : integer (deprecated)
  Returns an integer value corresponding to the number of bytes present in the
  request buffer. This is mostly used in ACL. It is important to understand
  that this test does not return false as long as the buffer is changing. This
  means that a check with equality to zero will almost always immediately match
  at the beginning of the session, while a test for more data will wait for
  that data to come in and return false only when haproxy is certain that no
  more data will come in. This test was designed to be used with TCP request
  content inspection.

req.payload(<offset>,<length>) : binary
  This extracts a binary block of <length> bytes and starting at byte <offset>
  in the request buffer. As a special case, if the <length> argument is zero,
  the the whole buffer from <offset> to the end is extracted. This can be used
  with ACLs in order to check for the presence of some content in a buffer at
  any location.

  ACL alternatives :
    payload(<offset>,<length>) : hex binary match

req.payload_lv(<offset1>,<length>[,<offset2>]) : binary
  This extracts a binary block whose size is specified at <offset1> for <length>
  bytes, and which starts at <offset2> if specified or just after the length in
  the request buffer. The <offset2> parameter also supports relative offsets if
  prepended with a '+' or '-' sign.

  ACL alternatives :
    payload_lv(<offset1>,<length>[,<offset2>]) : hex binary match

  Example : please consult the example from the "stick store-response" keyword.

req.proto_http : boolean
req_proto_http : boolean (deprecated)
  Returns true when data in the request buffer look like HTTP and correctly
  parses as such. It is the same parser as the common HTTP request parser which
  is used so there should be no surprises. The test does not match until the
  request is complete, failed or timed out. This test may be used to report the
  protocol in TCP logs, but the biggest use is to block TCP request analysis
  until a complete HTTP request is present in the buffer, for example to track
  a header.

  Example:
        # track request counts per "base" (concatenation of Host+URL)
        tcp-request inspect-delay 10s
        tcp-request content reject if !HTTP
        tcp-request content track-sc0 base table req-rate

req.rdp_cookie([<name>]) : string
rdp_cookie([<name>]) : string (deprecated)
  When the request buffer looks like the RDP protocol, extracts the RDP cookie
  <name>, or any cookie if unspecified. The parser only checks for the first
  cookie, as illustrated in the RDP protocol specification. The cookie name is
```



```
  case insensitive. Generally the "MSTS" cookie name will be used, as it can
  contain the user name of the client connecting to the server if properly
  configured on the client. The "MSTSHASH" cookie is often used as well for
  session stickiness to servers.

  This differs from "balance rdp-cookie" in that any balancing algorithm may be
  used and thus the distribution of clients to backend servers is not linked to
  a hash of the RDP cookie. It is envisaged that using a balancing algorithm
  such as "balance roundrobin" or "balance leastconn" will lead to a more even
  distribution of clients to backend servers than the hash used by "balance
  rdp-cookie".

  ACL derivatives :
    req_rdp_cookie([<name>]) : exact string match

  Example :
   listen tse-farm
       bind 0.0.0.0:3389
       # wait up to 5s for an RDP cookie in the request
       tcp-request inspect-delay 5s
       tcp-request content accept if RDP_COOKIE
       # apply RDP cookie persistence
       persist rdp-cookie
       # Persist based on the mstshash cookie
       # This is only useful makes sense if
       # balance rdp-cookie is not used
       stick-table type string size 204800
       stick on req.rdp_cookie(mstshash)
       server srv1 1.1.1.1:3389
       server srv1 1.1.1.2:3389

  See also : "balance rdp-cookie", "persist rdp-cookie", "tcp-request" and the
  "req_rdp_cookie" ACL.

req.rdp_cookie_cnt([name]) : integer
rdp_cookie_cnt([name]) : integer (deprecated)
  Tries to parse the request buffer as RDP protocol, then returns an integer
  corresponding to the number of RDP cookies found. If an optional cookie name
  is passed, only cookies matching this name are considered. This is mostly
  used in ACL.

  ACL derivatives :
    req_rdp_cookie_cnt([<name>]) : integer match

req.ssl_hello_type : integer
req_ssl_hello_type : integer (deprecated)
  Returns an integer value containing the type of the SSL hello message found
  in the request buffer if the buffer contains data that parse as a complete
  SSL (v3 or superior) client hello message. Note that this only applies to raw
  contents found in the request buffer and not to contents deciphered via an
  SSL data layer, so this will not work with "bind" lines having the "ssl"
  option. This is mostly used in ACL to detect presence of an SSL hello message
  that is supposed to contain an SSL session ID usable for stickiness.

req.ssl_sni : string
req_ssl_sni : string (deprecated)
  Returns a string containing the value of the Server Name TLS extension sent
  by a client in a TLS stream passing through the request buffer if the buffer
  contains data that parse as a complete SSL (v3 or superior) client hello
  message. Note that this only applies to raw contents found in the request
  buffer and not to contents deciphered via an SSL data layer, so this will not
  work with "bind" lines having the "ssl" option. SNI normally contains the
  name of the host the client tries to connect to (for recent browsers). SNI is
  useful for allowing or denying access to certain hosts when SSL/TLS is used
  by the client. This test was designed to be used with TCP request content
```

```
   inspection. If content switching is needed, it is recommended to first wait
   for a complete client hello (type 1), like in the example below. See also
   "ssl_fc_sni".

   ACL derivatives :
     req_ssl_sni : exact string match

   Examples :
      # Wait for a client hello for at most 5 seconds
      tcp-request inspect-delay 5s
      tcp-request content accept if { req_ssl_hello_type 1 }
      use_backend bk_allow if { req_ssl_sni -f allowed_sites }
      default_backend bk_sorry_page

 res.ssl_hello_type : integer
 rep_ssl_hello_type : integer (deprecated)
   Returns an integer value containing the type of the SSL hello message found
   in the response buffer if the buffer contains data that parses as a complete
   SSL (v3 or superior) hello message. Note that this only applies to raw
   contents found in the response buffer and not to contents deciphered via an
   SSL data layer, so this will not work with "server" lines having the "ssl"
   option. This is mostly used in ACL to detect presence of an SSL hello message
   that is supposed to contain an SSL session ID usable for stickiness.

 req.ssl_ver : integer
 req_ssl_ver : integer (deprecated)
   Returns an integer value containing the version of the SSL/TLS protocol of a
   stream present in the request buffer. Both SSLv2 hello messages and SSLv3
   messages are supported. TLSv1 is announced as SSL version 3.1. The value is
   composed of the major version multiplied by 65536, added to the minor
   version. Note that this only applies to raw contents found in the request
   buffer and not to contents deciphered via an SSL data layer, so this will not
   work with "bind" lines having the "ssl" option. The ACL version of the test
   matches against a decimal notation in the form MAJOR.MINOR (eg: 3.1). This
   fetch is mostly used in ACL.

   ACL derivatives :
     req_ssl_ver : decimal match

 res.len : integer
   Returns an integer value corresponding to the number of bytes present in the
   response buffer. This is mostly used in ACL. It is important to understand
   that this test does not return false as long as the buffer is changing. This
   means that a check with equality to zero will almost always immediately match
   at the beginning of the session, while a test for more data will wait for
   that data to come in and return false only when haproxy is certain that no
   more data will come in. This test was designed to be used with TCP response
   content inspection.

 res.payload(<offset>,<length>) : binary
   This extracts a binary block of <length> bytes and starting at byte <offset>
   in the response buffer. As a special case, if the <length> argument is zero,
   the the whole buffer from <offset> to the end is extracted. This can be used
   with ACLs in order to check for the presence of some content in a buffer at
   any location.

 res.payload_lv(<offset1>,<length>[,<offset2>]) : binary
   This extracts a binary block whose size is specified at <offset1> for <length>
   bytes, and which starts at <offset2> if specified or just after the length in
   the response buffer. The <offset2> parameter also supports relative offsets
   if prepended with a '+' or '-' sign.

   Example : please consult the example from the "stick store-response" keyword.

 wait_end : boolean
```



```
   This fetch either returns true when the inspection period is over, or does
   not fetch. It is only used in ACLs, in conjunction with content analysis to
   avoid returning a wrong verdict early.  It may also be used to delay some
   actions, such as a delayed reject for some special addresses. Since it either
   stops the rules evaluation or immediately returns true, it is recommended to
   use this acl as the last one in a rule.  Please note that the default ACL
   "WAIT_END" is always usable without prior declaration. This test was designed
   to be used with TCP request content inspection.

   Examples :
      # delay every incoming request by 2 seconds
      tcp-request inspect-delay 2s
      tcp-request content accept if WAIT_END

      # don't immediately tell bad guys they are rejected
      tcp-request inspect-delay 10s
      acl goodguys src 10.0.0.0/24
      acl badguys  src 10.0.1.0/24
      tcp-request content accept if goodguys
      tcp-request content reject if badguys WAIT_END
      tcp-request content reject


 7.3.6. Fetching HTTP samples (Layer 7)
 --------------------------------------

 It is possible to fetch samples from HTTP contents, requests and responses.
 This application layer is also called layer 7. It is only possible to fetch the
 data in this section when a full HTTP request or response has been parsed from
 its respective request or response buffer. This is always the case with all
 HTTP specific rules and for sections running with "mode http". When using TCP
 content inspection, it may be necessary to support an inspection delay in order
 to let the request or response come in first. These fetches may require a bit
 more CPU resources than the layer 4 ones, but not much since the request and
 response are indexed.

 base : string
   This returns the concatenation of the first Host header and the path part of
   the request, which starts at the first slash and ends before the question
   mark. It can be useful in virtual hosted environments to detect URL abuses as
   well as to improve shared caches efficiency. Using this with a limited size
   stick table also allows one to collect statistics about most commonly
   requested objects by host/path. With ACLs it can allow simple content
   switching rules involving the host and the path at the same time, such as
   "www.example.com/favicon.ico". See also "path" and "uri".

   ACL derivatives :
     base     : exact string match
     base_beg : prefix match
     base_dir : subdir match
     base_dom : domain match
     base_end : suffix match
     base_len : length match
     base_reg : regex match
     base_sub : substring match

 base32 : integer
   This returns a 32-bit hash of the value returned by the "base" fetch method
   above. This is useful to track per-URL activity on high traffic sites without
   having to store all URLs. Instead a shorter hash is stored, saving a lot of
   memory. The output type is an unsigned integer.

 base32+src : binary
   This returns the concatenation of the base32 fetch above and the src fetch
   below. The resulting type is of type binary, with a size of 8 or 20 bytes
```

```
11311    depending on the source address family. This can be used to track per-IP,
11312    per-URL counters.
11313
11314  capture.req.hdr(<idx>) : string
11315    This extracts the content of the header captured by the "capture request
11316    header", idx is the position of the capture keyword in the configuration.
11317    The first entry is an index of 0. See also: "capture request header".
11318
11319  capture.req.method : string
11320    This extracts the METHOD of an HTTP request. It can be used in both request
11321    and response. Unlike "method", it can be used in both request and response
11322    because it's allocated.
11323
11324  capture.req.uri : string
11325    This extracts the request's URI, which starts at the first slash and ends
11326    before the first space in the request (without the host part). Unlike "path"
11327    and "url", it can be used in both request and response because it's
11328    allocated.
11329
11330  capture.req.ver : string
11331    This extracts the request's HTTP version and returns either "HTTP/1.0" or
11332    "HTTP/1.1". Unlike "req.ver", it can be used in both request, response, and
11333    logs because it relies on a persistent flag.
11334
11335  capture.res.hdr(<idx>) : string
11336    This extracts the content of the header captured by the "capture response
11337    header", idx is the position of the capture keyword in the configuration.
11338    The first entry is an index of 0.
11339    See also: "capture response header"
11340
11341  capture.res.ver : string
11342    This extracts the response's HTTP version and returns either "HTTP/1.0" or
11343    "HTTP/1.1". Unlike "res.ver", it can be used in logs because it relies on a
11344    persistent flag.
11345
11346  req.cook([<name>]) : string
11347  cook([<name>]) : string (deprecated)
11348    This extracts the last occurrence of the cookie name <name> on a "Cookie"
11349    header line from the request, and returns its value as string. If no name is
11350    specified, the first cookie value is returned. When used with ACLs, all
11351    matching cookies are evaluated. Spaces around the name and the value are
11352    ignored as requested by the Cookie header specification (RFC6265). The cookie
11353    name is case-sensitive. Empty cookies are valid, so an empty cookie may very
11354    well return an empty value if it is present. Use the "found" match to detect
11355    presence. Use the res.cook() variant for response cookies sent by the server.
11356
11357    ACL derivatives :
11358      cook([<name>])     : exact string match
11359      cook_beg([<name>]) : prefix match
11360      cook_dir([<name>]) : subdir match
11361      cook_dom([<name>]) : domain match
11362      cook_end([<name>]) : suffix match
11363      cook_len([<name>]) : length match
11364      cook_reg([<name>]) : regex match
11365      cook_sub([<name>]) : substring match
11366
11367  req.cook_cnt([<name>]) : integer
11368  cook_cnt([<name>]) : integer (deprecated)
11369    Returns an integer value representing the number of occurrences of the cookie
11370    <name> in the request, or all cookies if <name> is not specified.
11371
11372  req.cook_val([<name>]) : integer
11373  cook_val([<name>]) : integer (deprecated)
11374    This extracts the last occurrence of the cookie name <name> on a "Cookie"
11375    header line from the request, and converts its value to an integer which is
```



```
11376    returned. If no name is specified, the first cookie value is returned. When
11377    used in ACLs, all matching names are iterated over until a value matches.
11378
11379  cookie([<name>]) : string (deprecated)
11380    This extracts the last occurrence of the cookie name <name> on a "Cookie"
11381    header line from the request, or a "Set-Cookie" header from the response, and
11382    returns its value as a string. A typical use is to get multiple clients
11383    sharing a same profile use the same server. This can be similar to what
11384    "appsession" does with the "request-learn" statement, but with support for
11385    multi-peer synchronization and state keeping across restarts. If no name is
11386    specified, the first cookie value is returned. This fetch should not be used
11387    anymore and should be replaced by req.cook() or res.cook() instead as it
11388    ambiguously uses the direction based on the context where it is used.
11389    See also : "appsession".
11390
11391  hdr([<name>[,<occ>]]) : string
11392    This is equivalent to req.hdr() when used on requests, and to res.hdr() when
11393    used on responses. Please refer to these respective fetches for more details.
11394    In case of doubt about the fetch direction, please use the explicit ones.
11395    Note that contrary to the hdr() sample fetch method, the hdr_* ACL keywords
11396    unambiguously apply to the request headers.
11397
11398  req.fhdr(<name>[,<occ>]) : string
11399    This extracts the last occurrence of header <name> in an HTTP request. When
11400    used from an ACL, all occurrences are iterated over until a match is found.
11401    Optionally, a specific occurrence might be specified as a position number.
11402    Positive values indicate a position from the first occurrence, with 1 being
11403    the first one. Negative values indicate positions relative to the last one,
11404    with -1 being the last one. It differs from req.hdr() in that any commas
11405    present in the value are returned and are not used as delimiters. This is
11406    sometimes useful with headers such as User-Agent.
11407
11408  req.fhdr_cnt([<name>]) : integer
11409    Returns an integer value representing the number of occurrences of request
11410    header field name <name>, or the total number of header fields if <name> is
11411    not specified. Contrary to its req.hdr_cnt() cousin, this function returns
11412    the number of full line headers and does not stop on commas.
11413
11414  req.hdr([<name>[,<occ>]]) : string
11415    This extracts the last occurrence of header <name> in an HTTP request. When
11416    used from an ACL, all occurrences are iterated over until a match is found.
11417    Optionally, a specific occurrence might be specified as a position number.
11418    Positive values indicate a position from the first occurrence, with 1 being
11419    the first one. Negative values indicate positions relative to the last one,
11420    with -1 being the last one. A typical use is with the X-Forwarded-For header
11421    once converted to IP, associated with an IP stick-table. The function
11422    considers any comma as a delimiter for distinct values. If full-line headers
11423    are desired instead, use req.fhdr(). Please carefully check RFC2616 to know
11424    how certain headers are supposed to be parsed. Also, some of them are case
11425    insensitive (eg: Connection).
11426
11427    ACL derivatives :
11428      hdr([<name>[,<occ>]])     : exact string match
11429      hdr_beg([<name>[,<occ>]]) : prefix match
11430      hdr_dir([<name>[,<occ>]]) : subdir match
11431      hdr_dom([<name>[,<occ>]]) : domain match
11432      hdr_end([<name>[,<occ>]]) : suffix match
11433      hdr_len([<name>[,<occ>]]) : length match
11434      hdr_reg([<name>[,<occ>]]) : regex match
11435      hdr_sub([<name>[,<occ>]]) : substring match
11436
11437  req.hdr_cnt([<name>]) : integer
11438  hdr_cnt([<header>]) : integer (deprecated)
11439    Returns an integer value representing the number of occurrences of request
11440    header field name <name>, or the total number of header field values if
```

```
11441    <name> is not specified. It is important to remember that one header line may
11442    count as several headers if it has several values. The function considers any
11443    comma as a delimiter for distinct values. If full-line headers are desired
11444    instead, req.fhdr_cnt() should be used instead. With ACLs, it can be used to
11445    detect presence, absence or abuse of a specific header, as well as to block
11446    request smuggling attacks by rejecting requests which contain more than one
11447    of certain headers. See "req.hdr" for more information on header matching.
11448
11449  req.hdr_ip([<name>[,<occ>]]) : ip
11450  hdr_ip([<name>[,<occ>]]) : ip (deprecated)
11451    This extracts the last occurrence of header <name> in an HTTP request,
11452    converts it to an IPv4 or IPv6 address and returns this address. When used
11453    with ACLs, all occurrences are checked, and if <name> is omitted, every value
11454    of every header is checked. Optionally, a specific occurrence might be
11455    specified as a position number. Positive values indicate a position from the
11456    first occurrence, with 1 being the first one.  Negative values indicate
11457    positions relative to the last one, with -1 being the last one. A typical use
11458    is with the X-Forwarded-For and X-Client-IP headers.
11459
11460  req.hdr_val([<name>[,<occ>]]) : integer
11461  hdr_val([<name>[,<occ>]]) : integer (deprecated)
11462    This extracts the last occurrence of header <name> in an HTTP request, and
11463    converts it to an integer value. When used with ACLs, all occurrences are
11464    checked, and if <name> is omitted, every value of every header is checked.
11465    Optionally, a specific occurrence might be specified as a position number.
11466    Positive values indicate a position from the first occurrence, with 1 being
11467    the first one. Negative values indicate positions relative to the last one,
11468    with -1 being the last one. A typical use is with the X-Forwarded-For header.
11469
11470  http_auth(<userlist>) : boolean
11471    Returns a boolean indicating whether the authentication data received from
11472    the client match a username & password stored in the specified userlist. This
11473    fetch function is not really useful outside of ACLs. Currently only http
11474    basic auth is supported.
11475
11476  http_auth_group(<userlist>) : string
11477    Returns a string corresponding to the user name found in the authentication
11478    data received from the client if both the user name and password are valid
11479    according to the specified userlist. The main purpose is to use it in ACLs
11480    where it is then checked whether the user belongs to any group within a list.
11481    This fetch function is not really useful outside of ACLs. Currently only http
11482    basic auth is supported.
11483
11484    ACL derivatives :
11485      http_auth_group(<userlist>) : group ...
11486      Returns true when the user extracted from the request and whose password is
11487      valid according to the specified userlist belongs to at least one of the
11488      groups.
11489
11490  http_first_req : boolean
11491    Returns true when the request being processed is the first one of the
11492    connection. This can be used to add or remove headers that may be missing
11493    from some requests when a request is not the first one, or to help grouping
11494    requests in the logs.
11495
11496  method : integer + string
11497    Returns an integer value corresponding to the method in the HTTP request. For
11498    example, "GET" equals 1 (check sources to establish the matching). Value 9
11499    means "other method" and may be converted to a string extracted from the
11500    stream. This should not be used directly as a sample, this is only meant to
11501    be used from ACLs, which transparently convert methods from patterns to these
11502    integer + string values. Some predefined ACL already check for most common
11503    methods.
11504
11505    ACL derivatives :
```



```
11506      method : case insensitive method match
11507
11508    Example :
11509        # only accept GET and HEAD requests
11510        acl valid_method method GET HEAD
11511        http-request deny if ! valid_method
11512
11513  path : string
11514    This extracts the request's URL path, which starts at the first slash and
11515    ends before the question mark (without the host part). A typical use is with
11516    prefetch-capable caches, and with portals which need to aggregate multiple
11517    information from databases and keep them in caches. Note that with outgoing
11518    caches, it would be wiser to use "url" instead. With ACLs, it's typically
11519    used to match exact file names (eg: "/login.php"), or directory parts using
11520    the derivative forms. See also the "url" and "base" fetch methods.
11521
11522    ACL derivatives :
11523      path     : exact string match
11524      path_beg : prefix match
11525      path_dir : subdir match
11526      path_dom : domain match
11527      path_end : suffix match
11528      path_len : length match
11529      path_reg : regex match
11530      path_sub : substring match
11531
11532  req.ver : string
11533  req_ver : string (deprecated)
11534    Returns the version string from the HTTP request, for example "1.1". This can
11535    be useful for logs, but is mostly there for ACL. Some predefined ACL already
11536    check for versions 1.0 and 1.1.
11537
11538    ACL derivatives :
11539      req_ver : exact string match
11540
11541  res.comp : boolean
11542    Returns the boolean "true" value if the response has been compressed by
11543    HAProxy, otherwise returns boolean "false". This may be used to add
11544    information in the logs.
11545
11546  res.comp_algo : string
11547    Returns a string containing the name of the algorithm used if the response
11548    was compressed by HAProxy, for example : "deflate". This may be used to add
11549    some information in the logs.
11550
11551  res.cook([<name>]) : string
11552  scook([<name>]) : string (deprecated)
11553    This extracts the last occurrence of the cookie name <name> on a "Set-Cookie"
11554    header line from the response, and returns its value as string. If no name is
11555    specified, the first cookie value is returned.
11556
11557    ACL derivatives :
11558      scook([<name>] : exact string match
11559
11560  res.cook_cnt([<name>]) : integer
11561  scook_cnt([<name>]) : integer (deprecated)
11562    Returns an integer value representing the number of occurrences of the cookie
11563    <name> in the response, or all cookies if <name> is not specified. This is
11564    mostly useful when combined with ACLs to detect suspicious responses.
11565
11566  res.cook_val([<name>]) : integer
11567  scook_val([<name>]) : integer (deprecated)
11568    This extracts the last occurrence of the cookie name <name> on a "Set-Cookie"
11569    header line from the response, and converts its value to an integer which is
11570    returned. If no name is specified, the first cookie value is returned.
```

```
11571
11572  res.fhdr([<name>[,<occ>]]) : string
11573    This extracts the last occurrence of header <name> in an HTTP response, or of
11574    the last header if no <name> is specified. Optionally, a specific occurrence
11575    might be specified as a position number. Positive values indicate a position
11576    from the first occurrence, with 1 being the first one. Negative values
11577    indicate positions relative to the last one, with -1 being the last one. It
11578    differs from res.hdr() in that any commas present in the value are returned
11579    and are not used as delimiters. If this is not desired, the res.hdr() fetch
11580    should be used instead. This is sometimes useful with headers such as Date or
11581    Expires.
11582
11583  res.fhdr_cnt([<name>]) : integer
11584    Returns an integer value representing the number of occurrences of response
11585    header field name <name>, or the total number of header fields if <name> is
11586    not specified. Contrary to its res.hdr_cnt() cousin, this function returns
11587    the number of full line headers and does not stop on commas. If this is not
11588    desired, the res.hdr_cnt() fetch should be used instead.
11589
11590  res.hdr([<name>[,<occ>]]) : string
11591  shdr([<name>[,<occ>]]) : string (deprecated)
11592    This extracts the last occurrence of header <name> in an HTTP response, or of
11593    the last header if no <name> is specified. Optionally, a specific occurrence
11594    might be specified as a position number. Positive values indicate a position
11595    from the first occurrence, with 1 being the first one. Negative values
11596    indicate positions relative to the last one, with -1 being the last one. This
11597    can be useful to learn some data into a stick-table. The function considers
11598    any comma as a delimiter for distinct values. If this is not desired, the
11599    res.fhdr() fetch should be used instead.
11600
11601    ACL derivatives :
11602      shdr([<name>[,<occ>]])     : exact string match
11603      shdr_beg([<name>[,<occ>]]) : prefix match
11604      shdr_dir([<name>[,<occ>]]) : subdir match
11605      shdr_dom([<name>[,<occ>]]) : domain match
11606      shdr_end([<name>[,<occ>]]) : suffix match
11607      shdr_len([<name>[,<occ>]]) : length match
11608      shdr_reg([<name>[,<occ>]]) : regex match
11609      shdr_sub([<name>[,<occ>]]) : substring match
11610
11611  res.hdr_cnt([<name>]) : integer
11612  shdr_cnt([<name>]) : integer (deprecated)
11613    Returns an integer value representing the number of occurrences of response
11614    header field name <name>, or the total number of header fields if <name> is
11615    not specified. The function considers any comma as a delimiter for distinct
11616    values. If this is not desired, the res.fhdr_cnt() fetch should be used
11617    instead.
11618
11619  res.hdr_ip([<name>[,<occ>]]) : ip
11620  shdr_ip([<name>[,<occ>]]) : ip (deprecated)
11621    This extracts the last occurrence of header <name> in an HTTP response,
11622    convert it to an IPv4 or IPv6 address and returns this address. Optionally, a
11623    specific occurrence might be specified as a position number. Positive values
11624    indicate a position from the first occurrence, with 1 being the first one.
11625    Negative values indicate positions relative to the last one, with -1 being
11626    the last one. This can be useful to learn some data into a stick table.
11627
11628  res.hdr_val([<name>[,<occ>]]) : integer
11629  shdr_val([<name>[,<occ>]]) : integer (deprecated)
11630    This extracts the last occurrence of header <name> in an HTTP response, and
11631    converts it to an integer value. Optionally, a specific occurrence might be
11632    specified as a position number. Positive values indicate a position from the
11633    first occurrence, with 1 being the first one. Negative values indicate
11634    positions relative to the last one, with -1 being the last one. This can be
11635    useful to learn some data into a stick table.
```



```
11636
11637  res.ver : string
11638  resp_ver : string (deprecated)
11639    Returns the version string from the HTTP response, for example "1.1". This
11640    can be useful for logs, but is mostly there for ACL.
11641
11642    ACL derivatives :
11643      resp_ver : exact string match
11644
11645  set-cookie([<name>]) : string (deprecated)
11646    This extracts the last occurrence of the cookie name <name> on a "Set-Cookie"
11647    header line from the response and uses the corresponding value to match. This
11648    can be comparable to what "appsession" does with default options, but with
11649    support for multi-peer synchronization and state keeping across restarts.
11650
11651    This fetch function is deprecated and has been superseded by the "res.cook"
11652    fetch. This keyword will disappear soon.
11653
11654    See also : "appsession"
11655
11656  status : integer
11657    Returns an integer containing the HTTP status code in the HTTP response, for
11658    example, 302. It is mostly used within ACLs and integer ranges, for example,
11659    to remove any Location header if the response is not a 3xx.
11660
11661  url : string
11662    This extracts the request's URL as presented in the request. A typical use is
11663    with prefetch-capable caches, and with portals which need to aggregate
11664    multiple information from databases and keep them in caches. With ACLs, using
11665    "path" is preferred over using "url", because clients may send a full URL as
11666    is normally done with proxies. The only real use is to match "*" which does
11667    not match in "path", and for which there is already a predefined ACL. See
11668    also "path" and "base".
11669
11670    ACL derivatives :
11671      url     : exact string match
11672      url_beg : prefix match
11673      url_dir : subdir match
11674      url_dom : domain match
11675      url_end : suffix match
11676      url_len : length match
11677      url_reg : regex match
11678      url_sub : substring match
11679
11680  url_ip : ip
11681    This extracts the IP address from the request's URL when the host part is
11682    presented as an IP address. Its use is very limited. For instance, a
11683    monitoring system might use this field as an alternative for the source IP in
11684    order to test what path a given source address would follow, or to force an
11685    entry in a table for a given source address. With ACLs it can be used to
11686    restrict access to certain systems through a proxy, for example when combined
11687    with option "http_proxy".
11688
11689  url_port : integer
11690    This extracts the port part from the request's URL. Note that if the port is
11691    not specified in the request, port 80 is assumed. With ACLs it can be used to
11692    restrict access to certain systems through a proxy, for example when combined
11693    with option "http_proxy".
11694
11695  urlp(<name>[,<delim>]) : string
11696  url_param(<name>[,<delim>]) : string
11697    This extracts the first occurrence of the parameter <name> in the query
11698    string, which begins after either '?' or <delim>, and which ends before '&',
11699    ';' or <delim>. The parameter name is case-sensitive. The result is a string
11700    corresponding to the value of the parameter <name> as presented in the
```

```
   request (no URL decoding is performed). This can be used for session
   stickiness based on a client ID, to extract an application cookie passed as a
   URL parameter, or in ACLs to apply some checks. Note that the ACL version of
   this fetch do not iterate over multiple parameters and stop at the first one
   as well.

   ACL derivatives :
     urlp(<name>[,<delim>])     : exact string match
     urlp_beg(<name>[,<delim>]) : prefix match
     urlp_dir(<name>[,<delim>]) : subdir match
     urlp_dom(<name>[,<delim>]) : domain match
     urlp_end(<name>[,<delim>]) : suffix match
     urlp_len(<name>[,<delim>]) : length match
     urlp_reg(<name>[,<delim>]) : regex match
     urlp_sub(<name>[,<delim>]) : substring match


   Example :
       # match http://example.com/foo?PHPSESSIONID=some_id
       stick on urlp(PHPSESSIONID)
       # match http://example.com/foo;JSESSIONID=some_id
       stick on urlp(JSESSIONID,;)

urlp_val(<name>[,<delim>]) : integer
  See "urlp" above. This one extracts the URL parameter <name> in the request
  and converts it to an integer value. This can be used for session stickiness
  based on a user ID for example, or with ACLs to match a page number or price.


7.4. Pre-defined ACLs
---------------------

Some predefined ACLs are hard-coded so that they do not have to be declared in
every frontend which needs them. They all have their names in upper case in
order to avoid confusion. Their equivalence is provided below.

ACL name          Equivalent to                Usage
---------------+----------------------------+---------------------------------
FALSE            always_false                 never match
HTTP             req_proto_http               match if protocol is valid HTTP
HTTP_1.0         req_ver 1.0                  match HTTP version 1.0
HTTP_1.1         req_ver 1.1                  match HTTP version 1.1
HTTP_CONTENT     hdr_val(content-length) gt 0 match an existing content-length
HTTP_URL_ABS     url_reg ^[^/:]*://           match absolute URL with scheme
HTTP_URL_SLASH   url_beg /                    match URL beginning with "/"
HTTP_URL_STAR    url     *                    match URL equal to "*"
LOCALHOST        src 127.0.0.1/8              match connection from local host
METH_CONNECT     method  CONNECT              match HTTP CONNECT method
METH_GET         method  GET HEAD             match HTTP GET or HEAD method
METH_HEAD        method  HEAD                 match HTTP HEAD method
METH_OPTIONS     method  OPTIONS              match HTTP OPTIONS method
METH_POST        method  POST                 match HTTP POST method
METH_TRACE       method  TRACE                match HTTP TRACE method
RDP_COOKIE       req_rdp_cookie_cnt gt 0      match presence of an RDP cookie
REQ_CONTENT      req_len gt 0                 match data in the request buffer
TRUE             always_true                  always match
WAIT_END         wait_end                     wait for end of content analysis
---------------+----------------------------+---------------------------------


8. Logging
----------

One of HAProxy's strong points certainly lies is its precise logs. It probably
provides the finest level of information available for such a product, which is
```



```
very important for troubleshooting complex environments. Standard information
provided in logs include client ports, TCP/HTTP state timers, precise session
state at termination and precise termination cause, information about decisions
to direct traffic to a server, and of course the ability to capture arbitrary
headers.

In order to improve administrators reactivity, it offers a great transparency
about encountered problems, both internal and external, and it is possible to
send logs to different sources at the same time with different level filters :

  - global process-level logs (system errors, start/stop, etc..)
  - per-instance system and internal errors (lack of resource, bugs, ...)
  - per-instance external troubles (servers up/down, max connections)
  - per-instance activity (client connections), either at the establishment or
    at the termination.

The ability to distribute different levels of logs to different log servers
allow several production teams to interact and to fix their problems as soon
as possible. For example, the system team might monitor system-wide errors,
while the application team might be monitoring the up/down for their servers in
real time, and the security team might analyze the activity logs with one hour
delay.


8.1. Log levels
---------------

TCP and HTTP connections can be logged with information such as the date, time,
source IP address, destination address, connection duration, response times,
HTTP request, HTTP return code, number of bytes transmitted, conditions
in which the session ended, and even exchanged cookies values. For example
track a particular user's problems. All messages may be sent to up to two
syslog servers. Check the "log" keyword in section 4.2 for more information
about log facilities.


8.2. Log formats
----------------

HAProxy supports 5 log formats. Several fields are common between these formats
and will be detailed in the following sections. A few of them may vary
slightly with the configuration, due to indicators specific to certain
options. The supported formats are as follows :

  - the default format, which is very basic and very rarely used. It only
    provides very basic information about the incoming connection at the moment
    it is accepted : source IP:port, destination IP:port, and frontend-name.
    This mode will eventually disappear so it will not be described to great
    extents.

  - the TCP format, which is more advanced. This format is enabled when "option
    tcplog" is set on the frontend. HAProxy will then usually wait for the
    connection to terminate before logging. This format provides much richer
    information, such as timers, connection counts, queue size, etc... This
    format is recommended for pure TCP proxies.

  - the HTTP format, which is the most advanced for HTTP proxying. This format
    is enabled when "option httplog" is set on the frontend. It provides the
    same information as the TCP format with some HTTP-specific fields such as
    the request, the status code, and captures of headers and cookies. This
    format is recommended for HTTP proxies.

  - the CLF HTTP format, which is equivalent to the HTTP format, but with the
    fields arranged in the same order as the CLF format. In this mode, all
    timers, captures, flags, etc... appear one per field after the end of the
```

```
11831      common fields, in the same order they appear in the standard HTTP format.
11832
11833    - the custom log format, allows you to make your own log line.
11834
11835  Next sections will go deeper into details for each of these formats. Format
11836  specification will be performed on a "field" basis. Unless stated otherwise, a
11837  field is a portion of text delimited by any number of spaces. Since syslog
11838  servers are susceptible of inserting fields at the beginning of a line, it is
11839  always assumed that the first field is the one containing the process name and
11840  identifier.
11841
11842  Note : Since log lines may be quite long, the log examples in sections below
11843         might be broken into multiple lines. The example log lines will be
11844         prefixed with 3 closing angle brackets ('>>>') and each time a log is
11845         broken into multiple lines, each non-final line will end with a
11846         backslash ('\') and the next line will start indented by two characters.
11847
11848
11849  8.2.1. Default log format
11850  -------------------------
11851
11852  This format is used when no specific option is set. The log is emitted as soon
11853  as the connection is accepted. One should note that this currently is the only
11854  format which logs the request's destination IP and ports.
11855
11856    Example :
11857          listen www
11858              mode http
11859              log global
11860              server srv1 127.0.0.1:8000
11861
11862      >>> Feb  6 12:12:09 localhost \
11863            haproxy[14385]: Connect from 10.0.1.2:33312 to 10.0.3.31:8012 \
11864            (www/HTTP)
11865
11866    Field   Format                                Extract from the example above
11867        1   process_name '[' pid ']:'                            haproxy[14385]:
11868        2   'Connect from'                                          Connect from
11869        3   source_ip ':' source_port                           10.0.1.2:33312
11870        4   'to'                                                            to
11871        5   destination_ip ':' destination_port                 10.0.3.31:8012
11872        6   '(' frontend_name '/' mode ')'                              (www/HTTP)
11873
11874  Detailed fields description :
11875    - "source_ip" is the IP address of the client which initiated the connection.
11876    - "source_port" is the TCP port of the client which initiated the connection.
11877    - "destination_ip" is the IP address the client connected to.
11878    - "destination_port" is the TCP port the client connected to.
11879    - "frontend_name" is the name of the frontend (or listener) which received
11880      and processed the connection.
11881    - "mode is the mode the frontend is operating (TCP or HTTP).
11882
11883  In case of a UNIX socket, the source and destination addresses are marked as
11884  "unix:" and the ports reflect the internal ID of the socket which accepted the
11885  connection (the same ID as reported in the stats).
11886
11887  It is advised not to use this deprecated format for newer installations as it
11888  will eventually disappear.
11889
11890
11891  8.2.2. TCP log format
11892  ---------------------
11893
11894  The TCP format is used when "option tcplog" is specified in the frontend, and
11895  is the recommended format for pure TCP proxies. It provides a lot of precious
```



```
11896  information for troubleshooting. Since this format includes timers and byte
11897  counts, the log is normally emitted at the end of the session. It can be
11898  emitted earlier if "option logasap" is specified, which makes sense in most
11899  environments with long sessions such as remote terminals. Sessions which match
11900  the "monitor" rules are never logged. It is also possible not to emit logs for
11901  sessions for which no data were exchanged between the client and the server, by
11902  specifying "option dontlognull" in the frontend. Successful connections will
11903  not be logged if "option dontlog-normal" is specified in the frontend. A few
11904  fields may slightly vary depending on some configuration options, those are
11905  marked with a star ('*') after the field name below.
11906
11907    Example :
11908          frontend fnt
11909              mode tcp
11910              option tcplog
11911              log global
11912              default_backend bck
11913
11914          backend bck
11915              server srv1 127.0.0.1:8000
11916
11917      >>> Feb  6 12:12:56 localhost \
11918            haproxy[14387]: 10.0.1.2:33313 [06/Feb/2009:12:12:51.443] fnt \
11919            bck/srv1 0/0/5007 212 -- 0/0/0/0/3 0/0
11920
11921    Field   Format                                Extract from the example above
11922        1   process_name '[' pid ']:'                            haproxy[14387]:
11923        2   client_ip ':' client_port                             10.0.1.2:33313
11924        3   '[' accept_date ']'                       [06/Feb/2009:12:12:51.443]
11925        4   frontend_name                                                   fnt
11926        5   backend_name '/' server_name                                bck/srv1
11927        6   Tw '/' Tc '/' Tt*                                          0/0/5007
11928        7   bytes_read*                                                     212
11929        8   termination_state                                                --
11930        9   actconn '/' feconn '/' beconn '/' srv_conn '/' retries*    0/0/0/0/3
11931       10   srv_queue '/' backend_queue                                     0/0
11932
11933  Detailed fields description :
11934    - "client_ip" is the IP address of the client which initiated the TCP
11935      connection to haproxy. If the connection was accepted on a UNIX socket
11936      instead, the IP address would be replaced with the word "unix". Note that
11937      when the connection is accepted on a socket configured with "accept-proxy"
11938      and the PROXY protocol is correctly used, then the logs will reflect the
11939      forwarded connection's information.
11940
11941    - "client_port" is the TCP port of the client which initiated the connection.
11942      If the connection was accepted on a UNIX socket instead, the port would be
11943      replaced with the ID of the accepting socket, which is also reported in the
11944      stats interface.
11945
11946    - "accept_date" is the exact date when the connection was received by haproxy
11947      (which might be very slightly different from the date observed on the
11948      network if there was some queuing in the system's backlog). This is usually
11949      the same date which may appear in any upstream firewall's log.
11950
11951    - "frontend_name" is the name of the frontend (or listener) which received
11952      and processed the connection.
11953
11954    - "backend_name" is the name of the backend (or listener) which was selected
11955      to manage the connection to the server. This will be the same as the
11956      frontend if no switching rule has been applied, which is common for TCP
11957      applications.
11958
11959    - "server_name" is the name of the last server to which the connection was
11960      sent, which might differ from the first one if there were connection errors
```

and a redispatch occurred. Note that this server belongs to the backend which processed the request. If the connection was aborted before reaching a server, "<NOSRV>" is indicated instead of a server name.

- "Tw" is the total time in milliseconds spent waiting in the various queues. It can be "-1" if the connection was aborted before reaching the queue. See "Timers" below for more details.

- "Tc" is the total time in milliseconds spent waiting for the connection to establish to the final server, including retries. It can be "-1" if the connection was aborted before a connection could be established. See "Timers" below for more details.

- "Tt" is the total time in milliseconds elapsed between the accept and the last close. It covers all possible processing. There is one exception, if "option logasap" was specified, then the time counting stops at the moment the log is emitted. In this case, a '+' sign is prepended before the value, indicating that the final one will be larger. See "Timers" below for more details.

- "bytes_read" is the total number of bytes transmitted from the server to the client when the log is emitted. If "option logasap" is specified, the this value will be prefixed with a '+' sign indicating that the final one may be larger. Please note that this value is a 64-bit counter, so log analysis tools must be able to handle it without overflowing.

- "termination_state" is the condition the session was in when the session ended. This indicates the session state, which side caused the end of session to happen, and for what reason (timeout, error, ...). The normal flags should be "--", indicating the session was closed by either end with no data remaining in buffers. See below "Session state at disconnection" for more details.

- "actconn" is the total number of concurrent connections on the process when the session was logged. It is useful to detect when some per-process system limits have been reached. For instance, if actconn is close to 512 when multiple connection errors occur, chances are high that the system limits the process to use a maximum of 1024 file descriptors and that all of them are used. See section 3 "Global parameters" to find how to tune the system.

- "feconn" is the total number of concurrent connections on the frontend when the session was logged. It is useful to estimate the amount of resource required to sustain high loads, and to detect when the frontend's "maxconn" has been reached. Most often when this value increases by huge jumps, it is because there is congestion on the backend servers, but sometimes it can be caused by a denial of service attack.

- "beconn" is the total number of concurrent connections handled by the backend when the session was logged. It includes the total number of concurrent connections active on servers as well as the number of connections pending in queues. It is useful to estimate the amount of additional servers needed to support high loads for a given application. Most often when this value increases by huge jumps, it is because there is congestion on the backend servers, but sometimes it can be caused by a denial of service attack.

- "srv_conn" is the total number of concurrent connections still active on the server when the session was logged. It can never exceed the server's configured "maxconn" parameter. If this value is very often close or equal to the server's "maxconn", it means that traffic regulation is involved a lot, meaning that either the server's maxconn value is too low, or that there aren't enough servers to process the load with an optimal response time. When only one of the server's "srv_conn" is high, it usually means that this server has some trouble causing the connections to take longer to be processed than on other servers.

- "retries" is the number of connection retries experienced by this session when trying to connect to the server. It must normally be zero, unless a server is being stopped at the same moment the connection was attempted. Frequent retries generally indicate either a network problem between haproxy and the server, or a misconfigured system backlog on the server preventing new connections from being queued. This field may optionally be prefixed with a '+' sign, indicating that the session has experienced a redispatch after the maximal retry count has been reached on the initial server. In this case, the server name appearing in the log is the one the connection was redispatched to, and not the first one, though both may sometimes be the same in case of hashing for instance. So as a general rule of thumb, when a '+' is present in front of the retry count, this count should not be attributed to the logged server.

- "srv_queue" is the total number of requests which were processed before this one in the server queue. It is zero when the request has not gone through the server queue. It makes it possible to estimate the approximate server's response time by dividing the time spent in queue by the number of requests in the queue. It is worth noting that if a session experiences a redispatch and passes through two server queues, their positions will be cumulated. A request should not pass through both the server queue and the backend queue unless a redispatch occurs.

- "backend_queue" is the total number of requests which were processed before this one in the backend's global queue. It is zero when the request has not gone through the global queue. It makes it possible to estimate the average queue length, which easily translates into a number of missing servers when divided by a server's "maxconn" parameter. It is worth noting that if a session experiences a redispatch, it may pass twice in the backend's queue, and then both positions will be cumulated. A request should not pass through both the server queue and the backend queue unless a redispatch occurs.

8.2.3. HTTP log format

The HTTP format is the most complete and the best suited for HTTP proxies. It is enabled by when "option httplog" is specified in the frontend. It provides the same level of information as the TCP format with additional features which are specific to the HTTP protocol. Just like the TCP format, the log is usually emitted at the end of the session, unless "option logasap" is specified, which generally only makes sense for download sites. A session which matches the "monitor" rules will never logged. It is also possible not to log sessions for which no data were sent by the client by specifying "option dontlognull" in the frontend. Successful connections will not be logged if "option dontlog-normal" is specified in the frontend.

Most fields are shared with the TCP log, some being different. A few fields may slightly vary depending on some configuration options. Those ones are marked with a star ('*') after the field name below.

Example :

```
      frontend http-in
          mode http
          option httplog
          log global
          default_backend bck

      backend static
          server srv1 127.0.0.1:8000
```

```
  >>> Feb  6 12:14:14 localhost \
        haproxy[14389]: 10.0.1.2:33317 [06/Feb/2009:12:14:14.655] http-in \
```

```
          static/srv1 10/0/30/69/109 200 2750 - - ---- 1/1/1/1/0 0/0 {1wt.eu} \
          {} "GET /index.html HTTP/1.1"

  Field   Format                                Extract from the example above
      1   process_name '[' pid ']:'                            haproxy[14389]:
      2   client_ip ':' client_port                             10.0.1.2:33317
      3   '[' accept_date ']'                     [06/Feb/2009:12:14:14.655]
      4   frontend_name                                             http-in
      5   backend_name '/' server_name                           static/srv1
      6   Tq '/' Tw '/' Tc '/' Tr '/' Tt*                     10/0/30/69/109
      7   status_code                                                   200
      8   bytes_read*                                                  2750
      9   captured_request_cookie                                         -
     10   captured_response_cookie                                        -
     11   termination_state                                            ----
     12   actconn '/' feconn '/' beconn '/' srv_conn '/' retries*    1/1/1/1/0
     13   srv_queue '/' backend_queue                                   0/0
     14   '{' captured_request_headers* '}'                  {haproxy.1wt.eu}
     15   '{' captured_response_headers* '}'                             {}
     16   '"' http_request '"'                     "GET /index.html HTTP/1.1"


Detailed fields description :
  - "client_ip" is the IP address of the client which initiated the TCP
    connection to haproxy. If the connection was accepted on a UNIX socket
    instead, the IP address would be replaced with the word "unix". Note that
    when the connection is accepted on a socket configured with "accept-proxy"
    and the PROXY protocol is correctly used, then the logs will reflect the
    forwarded connection's information.

  - "client_port" is the TCP port of the client which initiated the connection.
    If the connection was accepted on a UNIX socket instead, the port would be
    replaced with the ID of the accepting socket, which is also reported in the
    stats interface.

  - "accept_date" is the exact date when the TCP connection was received by
    haproxy (which might be very slightly different from the date observed on
    the network if there was some queuing in the system's backlog). This is
    usually the same date which may appear in any upstream firewall's log. This
    does not depend on the fact that the client has sent the request or not.

  - "frontend_name" is the name of the frontend (or listener) which received
    and processed the connection.

  - "backend_name" is the name of the backend (or listener) which was selected
    to manage the connection to the server. This will be the same as the
    frontend if no switching rule has been applied.

  - "server_name" is the name of the last server to which the connection was
    sent, which might differ from the first one if there were connection errors
    and a redispatch occurred. Note that this server belongs to the backend
    which processed the request. If the request was aborted before reaching a
    server, "<NOSRV>" is indicated instead of a server name. If the request was
    intercepted by the stats subsystem, "<STATS>" is indicated instead.

  - "Tq" is the total time in milliseconds spent waiting for the client to send
    a full HTTP request, not counting data. It can be "-1" if the connection
    was aborted before a complete request could be received. It should always
    be very small because a request generally fits in one single packet. Large
    times here generally indicate network trouble between the client and
    haproxy. See "Timers" below for more details.

  - "Tw" is the total time in milliseconds spent waiting in the various queues.
    It can be "-1" if the connection was aborted before reaching the queue.
    See "Timers" below for more details.
```



```

  - "Tc" is the total time in milliseconds spent waiting for the connection to
    establish to the final server, including retries. It can be "-1" if the
    request was aborted before a connection could be established. See "Timers"
    below for more details.

  - "Tr" is the total time in milliseconds spent waiting for the server to send
    a full HTTP response, not counting data. It can be "-1" if the request was
    aborted before a complete response could be received. It generally matches
    the server's processing time for the request, though it may be altered by
    the amount of data sent by the client to the server. Large times here on
    "GET" requests generally indicate an overloaded server. See "Timers" below
    for more details.

  - "Tt" is the total time in milliseconds elapsed between the accept and the
    last close. It covers all possible processing. There is one exception, if
    "option logasap" was specified, then the time counting stops at the moment
    the log is emitted. In this case, a '+' sign is prepended before the value,
    indicating that the final one will be larger. See "Timers" below for more
    details.

  - "status_code" is the HTTP status code returned to the client. This status
    is generally set by the server, but it might also be set by haproxy when
    the server cannot be reached or when its response is blocked by haproxy.

  - "bytes_read" is the total number of bytes transmitted to the client when
    the log is emitted. This does include HTTP headers. If "option logasap" is
    specified, the this value will be prefixed with a '+' sign indicating that
    the final one may be larger. Please note that this value is a 64-bit
    counter, so log analysis tools must be able to handle it without
    overflowing.

  - "captured_request_cookie" is an optional "name=value" entry indicating that
    the client had this cookie in the request. The cookie name and its maximum
    length are defined by the "capture cookie" statement in the frontend
    configuration. The field is a single dash ('-') when the option is not
    set. Only one cookie may be captured, it is generally used to track session
    ID exchanges between a client and a server to detect session crossing
    between clients due to application bugs. For more details, please consult
    the section "Capturing HTTP headers and cookies" below.

  - "captured_response_cookie" is an optional "name=value" entry indicating
    that the server has returned a cookie with its response. The cookie name
    and its maximum length are defined by the "capture cookie" statement in the
    frontend configuration. The field is a single dash ('-') when the option is
    not set. Only one cookie may be captured, it is generally used to track
    session ID exchanges between a client and a server to detect session
    crossing between clients due to application bugs. For more details, please
    consult the section "Capturing HTTP headers and cookies" below.

  - "termination_state" is the condition the session was in when the session
    ended. This indicates the session state, which side caused the end of
    session to happen, for what reason (timeout, error, ...), just like in TCP
    logs, and information about persistence operations on cookies in the last
    two characters. The normal flags should begin with "--", indicating the
    session was closed by either end with no data remaining in buffers. See
    below "Session state at disconnection" for more details.

  - "actconn" is the total number of concurrent connections on the process when
    the session was logged. It is useful to detect when some per-process system
    limits have been reached. For instance, if actconn is close to 512 or 1024
    when multiple connection errors occur, chances are high that the system
    limits the process to use a maximum of 1024 file descriptors and that all
    of them are used. See section 3 "Global parameters" to find how to tune the
    system.
```

- "feconn" is the total number of concurrent connections on the frontend when the session was logged. It is useful to estimate the amount of resource required to sustain high loads, and to detect when the frontend's "maxconn" has been reached. Most often when this value increases by huge jumps, it is because there is congestion on the backend servers, but sometimes it can be caused by a denial of service attack.

- "beconn" is the total number of concurrent connections handled by the backend when the session was logged. It includes the total number of concurrent connections active on servers as well as the number of connections pending in queues. It is useful to estimate the amount of additional servers needed to support high loads for a given application. Most often when this value increases by huge jumps, it is because there is congestion on the backend servers, but sometimes it can be caused by a denial of service attack.

- "srv_conn" is the total number of concurrent connections still active on the server when the session was logged. It can never exceed the server's configured "maxconn" parameter. If this value is very often close or equal to the server's "maxconn", it means that traffic regulation is involved a lot, meaning that either the server's maxconn value is too low, or that there aren't enough servers to process the load with an optimal response time. When only one of the server's "srv_conn" is high, it usually means that this server has some trouble causing the requests to take longer to be processed than on other servers.

- "retries" is the number of connection retries experienced by this session when trying to connect to the server. It must normally be zero, unless a server is being stopped at the same moment the connection was attempted. Frequent retries generally indicate either a network problem between haproxy and the server, or a misconfigured system backlog on the server preventing new connections from being queued. This field may optionally be prefixed with a '+' sign, indicating that the session has experienced a redispatch after the maximal retry count has been reached on the initial server. In this case, the server name appearing in the log is the one the connection was redispatched to, and not the first one, though both may sometimes be the same in case of hashing for instance. So as a general rule of thumb, when a '+' is present in front of the retry count, this count should not be attributed to the logged server.

- "srv_queue" is the total number of requests which were processed before this one in the server queue. It is zero when the request has not gone through the server queue. It makes it possible to estimate the approximate server's response time by dividing the time spent in queue by the number of requests in the queue. It is worth noting that if a session experiences a redispatch and passes through two server queues, their positions will be cumulated. A request should not pass through both the server queue and the backend queue unless a redispatch occurs.

- "backend_queue" is the total number of requests which were processed before this one in the backend's global queue. It is zero when the request has not gone through the global queue. It makes it possible to estimate the average queue length, which easily translates into a number of missing servers when divided by a server's "maxconn" parameter. It is worth noting that if a session experiences a redispatch, it may pass twice in the backend's queue, and then both positions will be cumulated. A request should not pass through both the server queue and the backend queue unless a redispatch occurs.

- "captured_request_headers" is a list of headers captured in the request due to the presence of the "capture request header" statement in the frontend. Multiple headers can be captured, they will be delimited by a vertical bar ('|'). When no capture is enabled, the braces do not appear, causing a shift of remaining fields. It is important to note that this field may

  contain spaces, and that using it requires a smarter log parser than when it's not used. Please consult the section "Capturing HTTP headers and cookies" below for more details.

- "captured_response_headers" is a list of headers captured in the response due to the presence of the "capture response header" statement in the frontend. Multiple headers can be captured, they will be delimited by a vertical bar ('|'). When no capture is enabled, the braces do not appear, causing a shift of remaining fields. It is important to note that this field may contain spaces, and that using it requires a smarter log parser than when it's not used. Please consult the section "Capturing HTTP headers and cookies" below for more details.

- "http_request" is the complete HTTP request line, including the method, request and HTTP version string. Non-printable characters are encoded (see below the section "Non-printable characters"). This is always the last field, and it is always delimited by quotes and is the only one which can contain quotes. If new fields are added to the log format, they will be added before this field. This field might be truncated if the request is huge and does not fit in the standard syslog buffer (1024 characters). This is the reason why this field must always remain the last one.

8.2.4. Custom log format

The directive log-format allows you to customize the logs in http mode and tcp mode. It takes a string as argument.

HAproxy understands some log format variables. % precedes log format variables. Variables can take arguments using braces ('{}'), and multiple arguments are separated by commas within the braces. Flags may be added or removed by prefixing them with a '+' or '-' sign.

Special variable "%o" may be used to propagate its flags to all other variables on the same format string. This is particularly handy with quoted string formats ("Q").

If a variable is named between square brackets ('[' .. ']') then it is used as a sample expression rule (see section 7.3). This it useful to add some less common information such as the client's SSL certificate's DN, or to log the key that would be used to store an entry into a stick table.

Note: spaces must be escaped. A space character is considered as a separator. In order to emit a verbatim '%', it must be preceded by another '%' resulting in '%%'. HAProxy will automatically merge consecutive separators.

Flags are :
  * Q: quote a string
  * X: hexadecimal representation (IPs, Ports, %Ts, %rt, %pid)

  Example:

    log-format %T\ %t\ Some\ Text
    log-format %{+Q}o\ %t\ %s\ %{-Q}r

At the moment, the default HTTP format is defined this way :

    log-format %ci:%cp\ [%t]\ %ft\ %b/%s\ %Tq/%Tw/%Tc/%Tr/%Tt\ %ST\ %B\ %CC\ \
                %CS\ %tsc\ %ac/%fc/%bc/%sc/%rc\ %sq/%bq\ %hr\ %hs\ %{+Q}r

the default CLF format is defined this way :

    log-format %{+Q}o\ %{-Q}ci\ -\ -\ [%T]\ %r\ %ST\ %B\ \"\"\ \"\"\ %cp\ \
                %ms\ %ft\ %b\ %s\ \%Tq\ %Tw\ %Tc\ %Tr\ %Tt\ %tsc\ %ac\ %fc\ \

```
                %bc\ %sc\ %rc\ %sq\ %bq\ %CC\ %CS\ \%hrl\ %hsl

and the default TCP format is defined this way :

    log-format %ci:%cp\ [%t]\ %ft\ %b/%s\ %Tw/%Tc/%Tt\ %B\ %ts\ \
               %ac/%fc/%bc/%sc/%rc\ %sq/%bq

Please refer to the table below for currently defined variables :

  +---+------+-----------------------------------------------+-------------+
  | R | var  | field name (8.2.2 and 8.2.3 for description)  | type        |
  +---+------+-----------------------------------------------+-------------+
  |   | %o   | special variable, apply flags on all next var |             |
  +---+------+-----------------------------------------------+-------------+
  |   | %B   | bytes_read           (from server to client)  | numeric     |
  | H | %CC  | captured_request_cookie                       | string      |
  | H | %CS  | captured_response_cookie                      | string      |
  |   | %H   | hostname                                      | string      |
  |   | %ID  | unique-id                                     | string      |
  |   | %ST  | status_code                                   | numeric     |
  |   | %T   | gmt_date_time                                 | date        |
  |   | %Tc  | Tc                                            | numeric     |
  |   | %Tl  | local_date_time                               | date        |
  | H | %Tq  | Tq                                            | numeric     |
  | H | %Tr  | Tr                                            | numeric     |
  |   | %Ts  | timestamp                                     | numeric     |
  |   | %Tt  | Tt                                            | numeric     |
  |   | %Tw  | Tw                                            | numeric     |
  |   | %U   | bytes_uploaded       (from client to server)  | numeric     |
  |   | %ac  | actconn                                       | numeric     |
  |   | %b   | backend_name                                  | string      |
  |   | %bc  | beconn      (backend concurrent connections)  | numeric     |
  |   | %bi  | backend_source_ip       (connecting address)  | IP          |
  |   | %bp  | backend_source_port     (connecting address)  | numeric     |
  |   | %bq  | backend_queue                                 | numeric     |
  |   | %ci  | client_ip                 (accepted address)  | IP          |
  |   | %cp  | client_port               (accepted address)  | numeric     |
  |   | %f   | frontend_name                                 | string      |
  |   | %fc  | feconn     (frontend concurrent connections)  | numeric     |
  |   | %fi  | frontend_ip             (accepting address)  | IP          |
  |   | %fp  | frontend_port           (accepting address)  | numeric     |
  |   | %ft  | frontend_name_transport ('~' suffix for SSL)  | string      |
  |   | %hr  | captured_request_headers default style        | string      |
  |   | %hrl | captured_request_headers CLF style            | string list |
  |   | %hs  | captured_response_headers default style       | string      |
  |   | %hsl | captured_response_headers CLF style           | string list |
  |   | %ms  | accept date milliseconds (left-padded with 0) | numeric     |
  |   | %pid | PID                                           | numeric     |
  | H | %r   | http_request                                  | string      |
  |   | %rc  | retries                                       | numeric     |
  |   | %rt  | request_counter (HTTP req or TCP session)     | numeric     |
  |   | %s   | server_name                                   | string      |
  |   | %sc  | srv_conn     (server concurrent connections)  | numeric     |
  |   | %si  | server_IP                 (target address)  | IP          |
  |   | %sp  | server_port               (target address)  | numeric     |
  |   | %sq  | srv_queue                                     | numeric     |
  | S | %sslc| ssl_ciphers (ex: AES-SHA)                     | string      |
  | S | %sslv| ssl_version (ex: TLSv1)                       | string      |
  |   | %t   | date_time      (with millisecond resolution)  | date        |
  |   | %ts  | termination_state                             | string      |
  | H | %tsc | termination_state with cookie status          | string      |
  +---+------+-----------------------------------------------+-------------+

    R = Restrictions : H = mode http only ; S = SSL only
```


8.2.5. Error log format

When an incoming connection fails due to an SSL handshake or an invalid PROXY protocol header, haproxy will log the event using a shorter, fixed line format. By default, logs are emitted at the LOG_INFO level, unless the option "log-separate-errors" is set in the backend, in which case the LOG_ERR level will be used. Connections on which no data are exchanged (eg: probes) are not logged if the "dontlognull" option is set.

The format looks like this :

```
    >>> Dec  3 18:27:14 localhost \
          haproxy[6103]: 127.0.0.1:56059 [03/Dec/2012:17:35:10.380] frt/f1: \
          Connection error during SSL handshake
```

Field	Format	Extract from the example above
1	process_name '[' pid ']:'	haproxy[6103]:
2	client_ip ':' client_port	127.0.0.1:56059
3	'[' accept_date ']'	[03/Dec/2012:17:35:10.380]
4	frontend_name "/" bind_name ":"	frt/f1:
5	message	Connection error during SSL handshake

These fields just provide minimal information to help debugging connection failures.

8.3. Advanced logging options

Some advanced logging options are often looked for but are not easy to find out just by looking at the various options. Here is an entry point for the few options which can enable better logging. Please refer to the keywords reference for more information about their usage.

8.3.1. Disabling logging of external tests

It is quite common to have some monitoring tools perform health checks on haproxy. Sometimes it will be a layer 3 load-balancer such as LVS or any commercial load-balancer, and sometimes it will simply be a more complete monitoring system such as Nagios. When the tests are very frequent, users often ask how to disable logging for those checks. There are three possibilities :

  - if connections come from everywhere and are just TCP probes, it is often
    desired to simply disable logging of connections without data exchange, by
    setting "option dontlognull" in the frontend. It also disables logging of
    port scans, which may or may not be desired.

  - if the connection come from a known source network, use "monitor-net" to
    declare this network as monitoring only. Any host in this network will then
    only be able to perform health checks, and their requests will not be
    logged. This is generally appropriate to designate a list of equipment
    such as other load-balancers.

  - if the tests are performed on a known URI, use "monitor-uri" to declare
    this URI as dedicated to monitoring. Any host sending this request will
    only get the result of a health-check, and the request will not be logged.

8.3.2. Logging before waiting for the session to terminate

The problem with logging at end of connection is that you have no clue about what is happening during very long sessions, such as remote terminal sessions or large file downloads. This problem can be worked around by specifying "option logasap" in the frontend. Haproxy will then log as soon as possible, just before data transfer begins. This means that in case of TCP, it will still log the connection status to the server, and in case of HTTP, it will log just after processing the server headers. In this case, the number of bytes reported is the number of header bytes sent to the client. In order to avoid confusion with normal logs, the total time field and the number of bytes are prefixed with a '+' sign which means that real numbers are certainly larger.

8.3.3. Raising log level upon errors

Sometimes it is more convenient to separate normal traffic from errors logs, for instance in order to ease error monitoring from log files. When the option "log-separate-errors" is used, connections which experience errors, timeouts, retries, redispatches or HTTP status codes 5xx will see their syslog level raised from "info" to "err". This will help a syslog daemon store the log in a separate file. It is very important to keep the errors in the normal traffic file too, so that log ordering is not altered. You should also be careful if you already have configured your syslog daemon to store all logs higher than "notice" in an "admin" file, because the "err" level is higher than "notice".

8.3.4. Disabling logging of successful connections

Although this may sound strange at first, some large sites have to deal with multiple thousands of logs per second and are experiencing difficulties keeping them intact for a long time or detecting errors within them. If the option "dontlog-normal" is set on the frontend, all normal connections will not be logged. In this regard, a normal connection is defined as one without any error, timeout, retry nor redispatch. In HTTP, the status code is checked too, and a response with a status 5xx is not considered normal and will be logged too. Of course, doing is is really discouraged as it will remove most of the useful information from the logs. Do this only if you have no other alternative.

8.4. Timing events

Timers provide a great help in troubleshooting network problems. All values are reported in milliseconds (ms). These timers should be used in conjunction with the session termination flags. In TCP mode with "option tcplog" set on the frontend, 3 control points are reported under the form "Tw/Tc/Tt", and in HTTP mode, 5 control points are reported under the form "Tq/Tw/Tc/Tr/Tt" :

  - Tq: total time to get the client request (HTTP mode only). It's the time elapsed between the moment the client connection was accepted and the moment the proxy received the last HTTP header. The value "-1" indicates that the end of headers (empty line) has never been seen. This happens when the client closes prematurely or times out.

  - Tw: total time spent in the queues waiting for a connection slot. It accounts for backend queue as well as the server queues, and depends on the queue size, and the time needed for the server to complete previous requests. The value "-1" means that the request was killed before reaching the queue, which is generally what happens with invalid or denied requests.

  - Tc: total time to establish the TCP connection to the server. It's the time elapsed between the moment the proxy sent the connection request, and the moment it was acknowledged by the server, or between the TCP SYN packet and

    the matching SYN/ACK packet in return. The value "-1" means that the connection never established.

  - Tr: server response time (HTTP mode only). It's the time elapsed between the moment the TCP connection was established to the server and the moment the server sent its complete response headers. It purely shows its request processing time, without the network overhead due to the data transmission. It is worth noting that when the client has data to send to the server, for instance during a POST request, the time already runs, and this can distort apparent response time. For this reason, it's generally wise not to trust too much this field for POST requests initiated from clients behind an untrusted network. A value of "-1" here means that the last the response header (empty line) was never seen, most likely because the server timeout stroke before the server managed to process the request.

  - Tt: total session duration time, between the moment the proxy accepted it and the moment both ends were closed. The exception is when the "logasap" option is specified. In this case, it only equals (Tq+Tw+Tc+Tr), and is prefixed with a '+' sign. From this field, we can deduce "Td", the data transmission time, by subtracting other timers when valid :

        Td = Tt - (Tq + Tw + Tc + Tr)

    Timers with "-1" values have to be excluded from this equation. In TCP mode, "Tq" and "Tr" have to be excluded too. Note that "Tt" can never be negative.

These timers provide precious indications on trouble causes. Since the TCP protocol defines retransmit delays of 3, 6, 12... seconds, we know for sure that timers close to multiples of 3s are nearly always related to lost packets due to network problems (wires, negotiation, congestion). Moreover, if "Tt" is close to a timeout value specified in the configuration, it often means that a session has been aborted on timeout.

Most common cases :

  - If "Tq" is close to 3000, a packet has probably been lost between the client and the proxy. This is very rare on local networks but might happen when clients are on far remote networks and send large requests. It may happen that values larger than usual appear here without any network cause. Sometimes, during an attack or just after a resource starvation has ended, haproxy may accept thousands of connections in a few milliseconds. The time spent accepting these connections will inevitably slightly delay processing of other connections, and it can happen that request times in the order of a few tens of milliseconds are measured after a few thousands of new connections have been accepted at once. Setting "option http-server-close" may display larger request times since "Tq" also measures the time spent waiting for additional requests.

  - If "Tc" is close to 3000, a packet has probably been lost between the server and the proxy during the server connection phase. This value should always be very low, such as 1 ms on local networks and less than a few tens of ms on remote networks.

  - If "Tr" is nearly always lower than 3000 except some rare values which seem to be the average majored by 3000, there are probably some packets lost between the proxy and the server.

  - If "Tt" is large even for small byte counts, it generally is because neither the client nor the server decides to close the connection, for instance because both have agreed on a keep-alive connection mode. In order to solve this issue, it will be needed to specify "option httpclose" on either the frontend or the backend. If the problem persists, it means that the server ignores the "close" connection mode and expects the client to close. Then it will be required to use "option forceclose". Having the

```
12611      smallest possible 'Tt' is important when connection regulation is used with
12612      the "maxconn" option on the servers, since no new connection will be sent
12613      to the server until another one is released.
12614
12615  Other noticeable HTTP log cases ('xx' means any value to be ignored) :
12616
12617    Tq/Tw/Tc/Tr/+Tt  The "option logasap" is present on the frontend and the log
12618                     was emitted before the data phase. All the timers are valid
12619                     except "Tt" which is shorter than reality.
12620
12621    -1/xx/xx/xx/Tt   The client was not able to send a complete request in time
12622                     or it aborted too early. Check the session termination flags
12623                     then "timeout http-request" and "timeout client" settings.
12624
12625    Tq/-1/xx/xx/Tt   It was not possible to process the request, maybe because
12626                     servers were out of order, because the request was invalid
12627                     or forbidden by ACL rules. Check the session termination
12628                     flags.
12629
12630    Tq/Tw/-1/xx/Tt   The connection could not establish on the server. Either it
12631                     actively refused it or it timed out after Tt-(Tq+Tw) ms.
12632                     Check the session termination flags, then check the
12633                     "timeout connect" setting. Note that the tarpit action might
12634                     return similar-looking patterns, with "Tw" equal to the time
12635                     the client connection was maintained open.
12636
12637    Tq/Tw/Tc/-1/Tt   The server has accepted the connection but did not return
12638                     a complete response in time, or it closed its connection
12639                     unexpectedly after Tt-(Tq+Tw+Tc) ms. Check the session
12640                     termination flags, then check the "timeout server" setting.
12641
12642
12643  8.5. Session state at disconnection
12644  -----------------------------------
12645
12646  TCP and HTTP logs provide a session termination indicator in the
12647  "termination_state" field, just before the number of active connections. It is
12648  2-characters long in TCP mode, and is extended to 4 characters in HTTP mode,
12649  each of which has a special meaning :
12650
12651    - On the first character, a code reporting the first event which caused the
12652      session to terminate :
12653
12654          C : the TCP session was unexpectedly aborted by the client.
12655
12656          S : the TCP session was unexpectedly aborted by the server, or the
12657              server explicitly refused it.
12658
12659          P : the session was prematurely aborted by the proxy, because of a
12660              connection limit enforcement, because a DENY filter was matched,
12661              because of a security check which detected and blocked a dangerous
12662              error in server response which might have caused information leak
12663              (eg: cacheable cookie).
12664
12665          L : the session was locally processed by haproxy and was not passed to
12666              a server. This is what happens for stats and redirects.
12667
12668          R : a resource on the proxy has been exhausted (memory, sockets, source
12669              ports, ...). Usually, this appears during the connection phase, and
12670              system logs should contain a copy of the precise error. If this
12671              happens, it must be considered as a very serious anomaly which
12672              should be fixed as soon as possible by any means.
12673
12674          I : an internal error was identified by the proxy during a self-check.
12675              This should NEVER happen, and you are encouraged to report any log
```



```
12676              containing this, because this would almost certainly be a bug. It
12677              would be wise to preventively restart the process after such an
12678              event too, in case it would be caused by memory corruption.
12679
12680          D : the session was killed by haproxy because the server was detected
12681              as down and was configured to kill all connections when going down.
12682
12683          U : the session was killed by haproxy on this backup server because an
12684              active server was detected as up and was configured to kill all
12685              backup connections when going up.
12686
12687          K : the session was actively killed by an admin operating on haproxy.
12688
12689          c : the client-side timeout expired while waiting for the client to
12690              send or receive data.
12691
12692          s : the server-side timeout expired while waiting for the server to
12693              send or receive data.
12694
12695          - : normal session completion, both the client and the server closed
12696              with nothing left in the buffers.
12697
12698    - on the second character, the TCP or HTTP session state when it was closed :
12699
12700          R : the proxy was waiting for a complete, valid REQUEST from the client
12701              (HTTP mode only). Nothing was sent to any server.
12702
12703          Q : the proxy was waiting in the QUEUE for a connection slot. This can
12704              only happen when servers have a 'maxconn' parameter set. It can
12705              also happen in the global queue after a redispatch consecutive to
12706              a failed attempt to connect to a dying server. If no redispatch is
12707              reported, then no connection attempt was made to any server.
12708
12709          C : the proxy was waiting for the CONNECTION to establish on the
12710              server. The server might at most have noticed a connection attempt.
12711
12712          H : the proxy was waiting for complete, valid response HEADERS from the
12713              server (HTTP only).
12714
12715          D : the session was in the DATA phase.
12716
12717          L : the proxy was still transmitting LAST data to the client while the
12718              server had already finished. This one is very rare as it can only
12719              happen when the client dies while receiving the last packets.
12720
12721          T : the request was tarpitted. It has been held open with the client
12722              during the whole "timeout tarpit" duration or until the client
12723              closed, both of which will be reported in the "Tw" timer.
12724
12725          - : normal session completion after end of data transfer.
12726
12727    - the third character tells whether the persistence cookie was provided by
12728      the client (only in HTTP mode) :
12729
12730          N : the client provided NO cookie. This is usually the case for new
12731              visitors, so counting the number of occurrences of this flag in the
12732              logs generally indicate a valid trend for the site frequentation.
12733
12734          I : the client provided an INVALID cookie matching no known server.
12735              This might be caused by a recent configuration change, mixed
12736              cookies between HTTP/HTTPS sites, persistence conditionally
12737              ignored, or an attack.
12738
12739          D : the client provided a cookie designating a server which was DOWN,
12740              so either "option persist" was used and the client was sent to
```

this server, or it was not set and the client was redispatched to
another server.

V : the client provided a VALID cookie, and was sent to the associated
server.

E : the client provided a valid cookie, but with a last date which was
older than what is allowed by the "maxidle" cookie parameter, so
the cookie is consider EXPIRED and is ignored. The request will be
redispatched just as if there was no cookie.

O : the client provided a valid cookie, but with a first date which was
older than what is allowed by the "maxlife" cookie parameter, so
the cookie is consider too OLD and is ignored. The request will be
redispatched just as if there was no cookie.

U : a cookie was present but was not used to select the server because
some other server selection mechanism was used instead (typically a
"use-server" rule).

- : does not apply (no cookie set in configuration).

- the last character reports what operations were performed on the persistence
cookie returned by the server (only in HTTP mode) :

N : NO cookie was provided by the server, and none was inserted either.

I : no cookie was provided by the server, and the proxy INSERTED one.
Note that in "cookie insert" mode, if the server provides a cookie,
it will still be overwritten and reported as "I" here.

U : the proxy UPDATED the last date in the cookie that was presented by
the client. This can only happen in insert mode with "maxidle". It
happens every time there is activity at a different date than the
date indicated in the cookie. If any other change happens, such as
a redispatch, then the cookie will be marked as inserted instead.

P : a cookie was PROVIDED by the server and transmitted as-is.

R : the cookie provided by the server was REWRITTEN by the proxy, which
happens in "cookie rewrite" or "cookie prefix" modes.

D : the cookie provided by the server was DELETED by the proxy.

- : does not apply (no cookie set in configuration).

The combination of the two first flags gives a lot of information about what
was happening when the session terminated, and why it did terminate. It can be
helpful to detect server saturation, network troubles, local system resource
starvation, attacks, etc...

The most common termination flags combinations are indicated below. They are
alphabetically sorted, with the lowercase set just after the upper case for
easier finding and understanding.

Flags	Reason
--	Normal termination.
CC	The client aborted before the connection could be established to the server. This can happen when haproxy tries to connect to a recently dead (or unchecked) server, and the client aborts while haproxy is waiting for the server to respond or for "timeout connect" to expire.
CD	The client unexpectedly aborted during data transfer. This can be

caused by a browser crash, by an intermediate equipment between the
client and haproxy which decided to actively break the connection,
by network routing issues between the client and haproxy, or by a
keep-alive session between the server and the client terminated first
by the client.

cD	The client did not send nor acknowledge any data for as long as the "timeout client" delay. This is often caused by network failures on the client side, or the client simply leaving the net uncleanly.
CH	The client aborted while waiting for the server to start responding. It might be the server taking too long to respond or the client clicking the 'Stop' button too fast.
cH	The "timeout client" stroke while waiting for client data during a POST request. This is sometimes caused by too large TCP MSS values for PPPoE networks which cannot transport full-sized packets. It can also happen when client timeout is smaller than server timeout and the server takes too long to respond.
CQ	The client aborted while its session was queued, waiting for a server with enough empty slots to accept it. It might be that either all the servers were saturated or that the assigned server was taking too long a time to respond.
CR	The client aborted before sending a full HTTP request. Most likely the request was typed by hand using a telnet client, and aborted too early. The HTTP status code is likely a 400 here. Sometimes this might also be caused by an IDS killing the connection between haproxy and the client. "option http-ignore-probes" can be used to ignore connections without any data transfer.
cR	The "timeout http-request" stroke before the client sent a full HTTP request. This is sometimes caused by too large TCP MSS values on the client side for PPPoE networks which cannot transport full-sized packets, or by clients sending requests by hand and not typing fast enough, or forgetting to enter the empty line at the end of the request. The HTTP status code is likely a 408 here. Note: recently, some browsers started to implement a "pre-connect" feature consisting in speculatively connecting to some recently visited web sites just in case the user would like to visit them. This results in many connections being established to web sites, which end up in 408 Request Timeout if the timeout strikes first, or 400 Bad Request when the browser decides to close them first. These ones pollute the log and feed the error counters. Some versions of some browsers have even been reported to display the error code. It is possible to work around the undesirable effects of this behaviour by adding "option http-ignore-probes" in the frontend, resulting in connections with zero data transfer to be totally ignored. This will definitely hide the errors of people experiencing connectivity issues though.
CT	The client aborted while its session was tarpitted. It is important to check if this happens on valid requests, in order to be sure that no wrong tarpit rules have been written. If a lot of them happen, it might make sense to lower the "timeout tarpit" value to something closer to the average reported "Tw" timer, in order not to consume resources for just a few attackers.
LR	The request was intercepted and locally handled by haproxy. Generally it means that this was a redirect or a stats request.
SC	The server or an equipment between it and haproxy explicitly refused the TCP connection (the proxy received a TCP RST or an ICMP message in return). Under some circumstances, it can also be the network stack telling the proxy that the server is unreachable (eg: no route,

or no ARP response on local network). When this happens in HTTP mode, the status code is likely a 502 or 503 here.

sC The "timeout connect" stroke before a connection to the server could complete. When this happens in HTTP mode, the status code is likely a 503 or 504 here.

SD The connection to the server died with an error during the data transfer. This usually means that haproxy has received an RST from the server or an ICMP message from an intermediate equipment while exchanging data with the server. This can be caused by a server crash or by a network issue on an intermediate equipment.

sD The server did not send nor acknowledge any data for as long as the "timeout server" setting during the data phase. This is often caused by too short timeouts on L4 equipments before the server (firewalls, load-balancers, ...), as well as keep-alive sessions maintained between the client and the server expiring first on haproxy.

SH The server aborted before sending its full HTTP response headers, or it crashed while processing the request. Since a server aborting at this moment is very rare, it would be wise to inspect its logs to control whether it crashed and why. The logged request may indicate a small set of faulty requests, demonstrating bugs in the application. Sometimes this might also be caused by an IDS killing the connection between haproxy and the server.

sH The "timeout server" stroke before the server could return its response headers. This is the most common anomaly, indicating too long transactions, probably caused by server or database saturation. The immediate workaround consists in increasing the "timeout server" setting, but it is important to keep in mind that the user experience will suffer from these long response times. The only long term solution is to fix the application.

sQ The session spent too much time in queue and has been expired. See the "timeout queue" and "timeout connect" settings to find out how to fix this if it happens too often. If it often happens massively in short periods, it may indicate general problems on the affected servers due to I/O or database congestion, or saturation caused by external attacks.

PC The proxy refused to establish a connection to the server because the process' socket limit has been reached while attempting to connect. The global "maxconn" parameter may be increased in the configuration so that it does not happen anymore. This status is very rare and might happen when the global "ulimit-n" parameter is forced by hand.

PD The proxy blocked an incorrectly formatted chunked encoded message in a request or a response, after the server has emitted its headers. In most cases, this will indicate an invalid message from the server to the client. Haproxy supports chunk sizes of up to 2GB - 1 (2147483647 bytes). Any larger size will be considered as an error.

PH The proxy blocked the server's response, because it was invalid, incomplete, dangerous (cache control), or matched a security filter. In any case, an HTTP 502 error is sent to the client. One possible cause for this error is an invalid syntax in an HTTP header name containing unauthorized characters. It is also possible but quite rare, that the proxy blocked a chunked-encoding request from the client due to an invalid syntax, before the server responded. In this case, an HTTP 400 error is sent to the client and reported in the logs.

PR The proxy blocked the client's HTTP request, either because of an

invalid HTTP syntax, in which case it returned an HTTP 400 error to the client, or because a deny filter matched, in which case it returned an HTTP 403 error.

PT The proxy blocked the client's request and has tarpitted its connection before returning it a 500 server error. Nothing was sent to the server. The connection was maintained open for as long as reported by the "Tw" timer field.

RC A local resource has been exhausted (memory, sockets, source ports) preventing the connection to the server from establishing. The error logs will tell precisely what was missing. This is very rare and can only be solved by proper system tuning.

The combination of the two last flags gives a lot of information about how persistence was handled by the client, the server and by haproxy. This is very important to troubleshoot disconnections, when users complain they have to re-authenticate. The commonly encountered flags are :

-- Persistence cookie is not enabled.

NN No cookie was provided by the client, none was inserted in the response. For instance, this can be in insert mode with "postonly" set on a GET request.

II A cookie designating an invalid server was provided by the client, a valid one was inserted in the response. This typically happens when a "server" entry is removed from the configuration, since its cookie value can be presented by a client when no other server knows it.

NI No cookie was provided by the client, one was inserted in the response. This typically happens for first requests from every user in "insert" mode, which makes it an easy way to count real users.

VN A cookie was provided by the client, none was inserted in the response. This happens for most responses for which the client has already got a cookie.

VU A cookie was provided by the client, with a last visit date which is not completely up-to-date, so an updated cookie was provided in response. This can also happen if there was no date at all, or if there was a date but the "maxidle" parameter was not set, so that the cookie can be switched to unlimited time.

EI A cookie was provided by the client, with a last visit date which is too old for the "maxidle" parameter, so the cookie was ignored and a new cookie was inserted in the response.

OI A cookie was provided by the client, with a first visit date which is too old for the "maxlife" parameter, so the cookie was ignored and a new cookie was inserted in the response.

DI The server designated by the cookie was down, a new server was selected and a new cookie was emitted in the response.

VI The server designated by the cookie was not marked dead but could not be reached. A redispatch happened and selected another one, which was then advertised in the response.

8.6. Non-printable characters

In order not to cause trouble to log analysis tools or terminals during log consulting, non-printable characters are not sent as-is into log files, but are

converted to the two-digits hexadecimal representation of their ASCII code, prefixed by the character '#'. The only characters that can be logged without being escaped are comprised between 32 and 126 (inclusive). Obviously, the escape character '#' itself is also encoded to avoid any ambiguity ("#23"). It is the same for the character '"' which becomes "#22", as well as '{', '|' and '}' when logging headers.

Note that the space character (' ') is not encoded in headers, which can cause issues for tools relying on space count to locate fields. A typical header containing spaces is "User-Agent".

Last, it has been observed that some syslog daemons such as syslog-ng escape the quote ('"') with a backslash ('\'). The reverse operation can safely be performed since no quote may appear anywhere else in the logs.

8.7. Capturing HTTP cookies

Cookie capture simplifies the tracking a complete user session. This can be achieved using the "capture cookie" statement in the frontend. Please refer to section 4.2 for more details. Only one cookie can be captured, and the same cookie will simultaneously be checked in the request ("Cookie:" header) and in the response ("Set-Cookie:" header). The respective values will be reported in the HTTP logs at the "captured_request_cookie" and "captured_response_cookie" locations (see section 8.2.3 about HTTP log format). When either cookie is not seen, a dash ('-') replaces the value. This way, it's easy to detect when a user switches to a new session for example, because the server will reassign it a new cookie. It is also possible to detect if a server unexpectedly sets a wrong cookie to a client, leading to session crossing.

  Examples :
```
        # capture the first cookie whose name starts with "ASPSESSION"
        capture cookie ASPSESSION len 32

        # capture the first cookie whose name is exactly "vgnvisitor"
        capture cookie vgnvisitor= len 32
```

8.8. Capturing HTTP headers

Header captures are useful to track unique request identifiers set by an upper proxy, virtual host names, user-agents, POST content-length, referrers, etc. In the response, one can search for information about the response length, how the server asked the cache to behave, or an object location during a redirection.

Header captures are performed using the "capture request header" and "capture response header" statements in the frontend. Please consult their definition in section 4.2 for more details.

It is possible to include both request headers and response headers at the same time. Non-existent headers are logged as empty strings, and if one header appears more than once, only its last occurrence will be logged. Request headers are grouped within braces '{' and '}' in the same order as they were declared, and delimited with a vertical bar '|' without any space. Response headers follow the same representation, but are displayed after a space following the request headers block. These blocks are displayed just before the HTTP request in the logs.

As a special case, it is possible to specify an HTTP header capture in a TCP frontend. The purpose is to enable logging of headers which will be parsed in an HTTP backend if the request is then switched to this HTTP backend.

  Example :


```
        # This instance chains to the outgoing proxy
        listen proxy-out
            mode http
            option httplog
            option logasap
            log global
            server cache1 192.168.1.1:3128

            # log the name of the virtual server
            capture request  header Host len 20

            # log the amount of data uploaded during a POST
            capture request  header Content-Length len 10

            # log the beginning of the referrer
            capture request  header Referer len 20

            # server name (useful for outgoing proxies only)
            capture response header Server len 20

            # logging the content-length is useful with "option logasap"
            capture response header Content-Length len 10

            # log the expected cache behaviour on the response
            capture response header Cache-Control len 8

            # the Via header will report the next proxy's name
            capture response header Via len 20

            # log the URL location during a redirection
            capture response header Location len 20

    >>> Aug  9 20:26:09 localhost \
          haproxy[2022]: 127.0.0.1:34014 [09/Aug/2004:20:26:09] proxy-out \
          proxy-out/cache1 0/0/0/162/+162 200 +350 - - ---- 0/0/0/0/0 0/0 \
          {fr.adserver.yahoo.co||http://fr.f416.mail.} {|864|private||} \
          "GET http://fr.adserver.yahoo.com/"

    >>> Aug  9 20:30:46 localhost \
          haproxy[2022]: 127.0.0.1:34020 [09/Aug/2004:20:30:46] proxy-out \
          proxy-out/cache1 0/0/0/182/+182 200 +279 - - ---- 0/0/0/0/0 0/0 \
          {w.ods.org||} {Formilux/0.1.8|3495|||} \
          "GET http://trafic.1wt.eu/ HTTP/1.1"

    >>> Aug  9 20:30:46 localhost \
          haproxy[2022]: 127.0.0.1:34028 [09/Aug/2004:20:30:46] proxy-out \
          proxy-out/cache1 0/0/2/126/+128 301 +223 - - ---- 0/0/0/0/0 0/0 \
          {www.sytadin.equipement.gouv.fr||http://trafic.1wt.eu/} \
          {Apache|230|||http://www.sytadin.} \
          "GET http://www.sytadin.equipement.gouv.fr/ HTTP/1.1"
```

8.9. Examples of logs

These are real-world examples of logs accompanied with an explanation. Some of them have been made up by hand. The syslog part has been removed for better reading. Their sole purpose is to explain how to decipher them.

```
    >>> haproxy[674]: 127.0.0.1:33318 [15/Oct/2003:08:31:57.130] px-http \
          px-http/srv1 6559/0/7/147/6723 200 243 - - ---- 5/3/3/1/0 0/0 \
          "HEAD / HTTP/1.0"

    => long request (6.5s) entered by hand through 'telnet'. The server replied
       in 147 ms, and the session ended normally ('----')
```

```
13131
13132      >>> haproxy[674]: 127.0.0.1:33319 [15/Oct/2003:08:31:57.149] px-http \
13133            px-http/srv1 6559/1230/7/147/6870 200 243 - - ---- 324/239/239/99/0 \
13134            0/9 "HEAD / HTTP/1.0"
13135
13136      => Idem, but the request was queued in the global queue behind 9 other
13137         requests, and waited there for 1230 ms.
13138
13139      >>> haproxy[674]: 127.0.0.1:33320 [15/Oct/2003:08:32:17.654] px-http \
13140            px-http/srv1 9/0/7/14/+30 200 +243 - - ---- 3/3/3/1/0 0/0 \
13141            "GET /image.iso HTTP/1.0"
13142
13143      => request for a long data transfer. The "logasap" option was specified, so
13144         the log was produced just before transferring data. The server replied in
13145         14 ms, 243 bytes of headers were sent to the client, and total time from
13146         accept to first data byte is 30 ms.
13147
13148      >>> haproxy[674]: 127.0.0.1:33320 [15/Oct/2003:08:32:17.925] px-http \
13149            px-http/srv1 9/0/7/14/30 502 243 - - PH-- 3/2/2/0/0 0/0 \
13150            "GET /cgi-bin/bug.cgi? HTTP/1.0"
13151
13152      => the proxy blocked a server response either because of an "rspdeny" or
13153         "rspideny" filter, or because the response was improperly formatted and
13154         not HTTP-compliant, or because it blocked sensitive information which
13155         risked being cached. In this case, the response is replaced with a "502
13156         bad gateway". The flags ("PH--") tell us that it was haproxy who decided
13157         to return the 502 and not the server.
13158
13159      >>> haproxy[18113]: 127.0.0.1:34548 [15/Oct/2003:15:18:55.798] px-http \
13160            px-http/<NOSRV> -1/-1/-1/-1/8490 -1 0 - - CR-- 2/2/2/0/0 0/0 ""
13161
13162      => the client never completed its request and aborted itself ("C---") after
13163         8.5s, while the proxy was waiting for the request headers ("-R--").
13164         Nothing was sent to any server.
13165
13166      >>> haproxy[18113]: 127.0.0.1:34549 [15/Oct/2003:15:19:06.103] px-http \
13167           px-http/<NOSRV> -1/-1/-1/-1/50001 408 0 - - cR-- 2/2/2/0/0 0/0 ""
13168
13169      => The client never completed its request, which was aborted by the
13170         time-out ("c---") after 50s, while the proxy was waiting for the request
13171         headers ("-R--").  Nothing was sent to any server, but the proxy could
13172         send a 408 return code to the client.
13173
13174      >>> haproxy[18989]: 127.0.0.1:34550 [15/Oct/2003:15:24:28.312] px-tcp \
13175            px-tcp/srv1 0/0/5007 0 cD 0/0/0/0/0 0/0
13176
13177      => This log was produced with "option tcplog". The client timed out after
13178         5 seconds ("c----").
13179
13180      >>> haproxy[18989]: 10.0.0.1:34552 [15/Oct/2003:15:26:31.462] px-http \
13181            px-http/srv1 3183/-1/-1/-1/11215 503 0 - - SC-- 205/202/202/115/3 \
13182            0/0 "HEAD / HTTP/1.0"
13183
13184      => The request took 3s to complete (probably a network problem), and the
13185         connection to the server failed ('SC--') after 4 attempts of 2 seconds
13186         (config says 'retries 3'), and no redispatch (otherwise we would have
13187         seen "/+3"). Status code 503 was returned to the client. There were 115
13188         connections on this server, 202 connections on this proxy, and 205 on
13189         the global process. It is possible that the server refused the
13190         connection because of too many already established.
13191
13192
13193   9. Statistics and monitoring
13194   ----------------------------
13195
```



```
13196   It is possible to query HAProxy about its status. The most commonly used
13197   mechanism is the HTTP statistics page. This page also exposes an alternative
13198   CSV output format for monitoring tools. The same format is provided on the
13199   Unix socket.
13200
13201
13202   9.1. CSV format
13203   ---------------
13204
13205   The statistics may be consulted either from the unix socket or from the HTTP
13206   page. Both means provide a CSV format whose fields follow. The first line
13207   begins with a sharp ('#') and has one word per comma-delimited field which
13208   represents the title of the column. All other lines starting at the second one
13209   use a classical CSV format using a comma as the delimiter, and the double quote
13210   ('"') as an optional text delimiter, but only if the enclosed text is ambiguous
13211   (if it contains a quote or a comma). The double-quote character ('"') in the
13212   text is doubled ('""'), which is the format that most tools recognize. Please
13213   do not insert any column before these ones in order not to break tools which
13214   use hard-coded column positions.
13215
13216   In brackets after each field name are the types which may have a value for
13217   that field. The types are L (Listeners), F (Frontends), B (Backends), and
13218   S (Servers).
13219
13220     0. pxname [LFBS]: proxy name
13221     1. svname [LFBS]: service name (FRONTEND for frontend, BACKEND for backend,
13222        any name for server/listener)
13223     2. qcur [..BS]: current queued requests. For the backend this reports the
13224        number queued without a server assigned.
13225     3. qmax [..BS]: max value of qcur
13226     4. scur [LFBS]: current sessions
13227     5. smax [LFBS]: max sessions
13228     6. slim [LFBS]: configured session limit
13229     7. stot [LFBS]: cumulative number of connections
13230     8. bin [LFBS]: bytes in
13231     9. bout [LFBS]: bytes out
13232    10. dreq [LFB.]: requests denied because of security concerns.
13233        - For tcp this is because of a matched tcp-request content rule.
13234        - For http this is because of a matched http-request or tarpit rule.
13235    11. dresp [LFBS]: responses denied because of security concerns.
13236        - For http this is because of a matched http-request rule, or
13237          "option checkcache".
13238    12. ereq [LF..]: request errors. Some of the possible causes are:
13239        - early termination from the client, before the request has been sent.
13240        - read error from the client
13241        - client timeout
13242        - client closed connection
13243        - various bad requests from the client.
13244        - request was tarpitted.
13245    13. econ [..BS]: number of requests that encountered an error trying to
13246        connect to a backend server. The backend stat is the sum of the stat
13247        for all servers of that backend, plus any connection errors not
13248        associated with a particular server (such as the backend having no
13249        active servers).
13250    14. eresp [..BS]: response errors. srv_abrt will be counted here also.
13251        Some other errors are:
13252        - write error on the client socket (won't be counted for the server stat)
13253        - failure applying filters to the response.
13254    15. wretr [..BS]: number of times a connection to a server was retried.
13255    16. wredis [..BS]: number of times a request was redispatched to another
13256        server. The server value counts the number of times that server was
13257        switched away from.
13258    17. status [LFBS]: status (UP/DOWN/NOLB/MAINT/MAINT(via)...)
13259    18. weight [..BS]: total weight (backend), server weight (server)
13260    19. act [..BS]: number of active servers (backend), server is active (server)
```

20. bck [..BS]: number of backup servers (backend), server is backup (server)
21. chkfail [...S]: number of failed checks. (Only counts checks failed when
    the server is up.)
22. chkdown [..BS]: number of UP->DOWN transitions. The backend counter counts
    transitions to the whole backend being down, rather than the sum of the
    counters for each server.
23. lastchg [..BS]: number of seconds since the last UP<->DOWN transition
24. downtime [..BS]: total downtime (in seconds). The value for the backend
    is the downtime for the whole backend, not the sum of the server downtime.
25. qlimit [...S]: configured maxqueue for the server, or nothing in the
    value is 0 (default, meaning no limit)
26. pid [LFBS]: process id (0 for first instance, 1 for second, ...)
27. iid [LFBS]: unique proxy id
28. sid [L..S]: server id (unique inside a proxy)
29. throttle [...S]: current throttle percentage for the server, when
    slowstart is active, or no value if not in slowstart.
30. lbtot [..BS]: total number of times a server was selected, either for new
    sessions, or when re-dispatching. The server counter is the number
    of times that server was selected.
31. tracked [...S]: id of proxy/server if tracking is enabled.
32. type [LFBS]: (0=frontend, 1=backend, 2=server, 3=socket/listener)
33. rate [.FBS]: number of sessions per second over last elapsed second
34. rate_lim [.F..]: configured limit on new sessions per second
35. rate_max [.FBS]: max number of new sessions per second
36. check_status [...S]: status of last health check, one of:
        UNK   -> unknown
        INI   -> initializing
        SOCKERR -> socket error
        L4OK   -> check passed on layer 4, no upper layers testing enabled
        L4TOUT -> layer 1-4 timeout
        L4CON   -> layer 1-4 connection problem, for example
                   "Connection refused" (tcp rst) or "No route to host" (icmp)
        L6OK   -> check passed on layer 6
        L6TOUT -> layer 6 (SSL) timeout
        L6RSP   -> layer 6 invalid response - protocol error
        L7OK   -> check passed on layer 7
        L7OKC   -> check conditionally passed on layer 7, for example 404 with
                   disable-on-404
        L7TOUT -> layer 7 (HTTP/SMTP) timeout
        L7RSP   -> layer 7 invalid response - protocol error
        L7STS   -> layer 7 response error, for example HTTP 5xx
37. check_code [...S]: layer5-7 code, if available
38. check_duration [...S]: time in ms took to finish last health check
39. hrsp_1xx [.FBS]: http responses with 1xx code
40. hrsp_2xx [.FBS]: http responses with 2xx code
41. hrsp_3xx [.FBS]: http responses with 3xx code
42. hrsp_4xx [.FBS]: http responses with 4xx code
43. hrsp_5xx [.FBS]: http responses with 5xx code
44. hrsp_other [.FBS]: http responses with other codes (protocol error)
45. hanafail [...S]: failed health checks details
46. req_rate [.F..]: HTTP requests per second over last elapsed second
47. req_rate_max [.F..]: max number of HTTP requests per second observed
48. req_tot [.F..]: total number of HTTP requests received
49. cli_abrt [..BS]: number of data transfers aborted by the client
50. srv_abrt [..BS]: number of data transfers aborted by the server
    (inc. in eresp)
51. comp_in [.FB.]: number of HTTP response bytes fed to the compressor
52. comp_out [.FB.]: number of HTTP response bytes emitted by the compressor
53. comp_byp [.FB.]: number of bytes that bypassed the HTTP compressor
    (CPU/BW limit)
54. comp_rsp [.FB.]: number of HTTP responses that were compressed
55. lastsess [..BS]: number of seconds since last session assigned to
    server/backend
56. last_chk [...S]: last health check contents or textual error
57. last_agt [...S]: last agent check contents or textual error

58. qtime [..BS]: the average queue time in ms over the 1024 last requests
59. ctime [..BS]: the average connect time in ms over the 1024 last requests
60. rtime [..BS]: the average response time in ms over the 1024 last requests
    (0 for TCP)
61. ttime [..BS]: the average total session time in ms over the 1024 last
    requests

9.2. Unix Socket commands

The stats socket is not enabled by default. In order to enable it, it is
necessary to add one line in the global section of the haproxy configuration.
A second line is recommended to set a larger timeout, always appreciated when
issuing commands by hand :

```
    global
        stats socket /var/run/haproxy.sock mode 600 level admin
        stats timeout 2m
```

It is also possible to add multiple instances of the stats socket by repeating
the line, and make them listen to a TCP port instead of a UNIX socket. This is
never done by default because this is dangerous, but can be handy in some
situations :

```
    global
        stats socket /var/run/haproxy.sock mode 600 level admin
        stats socket ipv4@192.168.0.1:9999 level admin
        stats timeout 2m
```

To access the socket, an external utility such as "socat" is required. Socat is a
swiss-army knife to connect anything to anything. We use it to connect terminals
to the socket, or a couple of stdin/stdout pipes to it for scripts. The two main
syntaxes we'll use are the following :

```
    # socat /var/run/haproxy.sock stdio
    # socat /var/run/haproxy.sock readline
```

The first one is used with scripts. It is possible to send the output of a
script to haproxy, and pass haproxy's output to another script. That's useful
for retrieving counters or attack traces for example.

The second one is only useful for issuing commands by hand. It has the benefit
that the terminal is handled by the readline library which supports line
editing and history, which is very convenient when issuing repeated commands
(eg: watch a counter).

The socket supports two operation modes :
  - interactive
  - non-interactive

The non-interactive mode is the default when socat connects to the socket. In
this mode, a single line may be sent. It is processed as a whole, responses are
sent back, and the connection closes after the end of the response. This is the
mode that scripts and monitoring tools use. It is possible to send multiple
commands in this mode, they need to be delimited by a semi-colon (';'). For
example :

```
    # echo "show info;show stat;show table" | socat /var/run/haproxy stdio
```

The interactive mode displays a prompt ('>') and waits for commands to be
entered on the line, then processes them, and displays the prompt again to wait
for a new command. This mode is entered via the "prompt" command which must be
sent on the first line in non-interactive mode. The mode is a flip switch, if
"prompt" is sent in interactive mode, it is disabled and the connection closes

after processing the last command of the same line.

For this reason, when debugging by hand, it's quite common to start with the "prompt" command :

```
   # socat /var/run/haproxy readline
   prompt
   > show info
   ...
   >
```

Since multiple commands may be issued at once, haproxy uses the empty line as a delimiter to mark an end of output for each command, and takes care of ensuring that no command can emit an empty line on output. A script can thus easily parse the output even when multiple commands were pipelined on a single line.

It is important to understand that when multiple haproxy processes are started on the same sockets, any process may pick up the request and will output its own stats.

The list of commands currently supported on the stats socket is provided below. If an unknown command is sent, haproxy displays the usage message which reminds all supported commands. Some commands support a more complex syntax, generally it will explain what part of the command is invalid when this happens.

add acl <acl> <pattern>
  Add an entry into the acl <acl>. <acl> is the #<id> or the <file> returned by "show acl". This command does not verify if the entry already exists. This command cannot be used if the reference <acl> is a file also used with a map. In this case, you must use the command "add map" in place of "add acl".

add map <map> <key> <value>
  Add an entry into the map <map> to associate the value <value> to the key <key>. This command does not verify if the entry already exists. It is mainly used to fill a map after a clear operation. Note that if the reference <map> is a file and is shared with a map, this map will contain also a new pattern entry.

clear counters
  Clear the max values of the statistics counters in each proxy (frontend & backend) and in each server. The cumulated counters are not affected. This can be used to get clean counters after an incident, without having to restart nor to clear traffic counters. This command is restricted and can only be issued on sockets configured for levels "operator" or "admin".

clear counters all
  Clear all statistics counters in each proxy (frontend & backend) and in each server. This has the same effect as restarting. This command is restricted and can only be issued on sockets configured for level "admin".

clear acl <acl>
  Remove all entries from the acl <acl>. <acl> is the #<id> or the <file> returned by "show acl". Note that if the reference <acl> is a file and is shared with a map, this map will be also cleared.

clear map <map>
  Remove all entries from the map <map>. <map> is the #<id> or the <file> returned by "show map". Note that if the reference <map> is a file and is shared with a acl, this acl will be also cleared.

clear table <table> [data.<type> <operator> <value>] | [key <key>]
  Remove entries from the stick-table <table>.

  This is typically used to unblock some users complaining they have been abusively denied access to a service, but this can also be used to clear some

  stickiness entries matching a server that is going to be replaced (see "show table" below for details). Note that sometimes, removal of an entry will be refused because it is currently tracked by a session. Retrying a few seconds later after the session ends is usual enough.

  In the case where no options arguments are given all entries will be removed.

  When the "data." form is used entries matching a filter applied using the stored data (see "stick-table" in section 4.2) are removed. A stored data type must be specified in <type>, and this data type must be stored in the table otherwise an error is reported. The data is compared according to <operator> with the 64-bit integer <value>. Operators are the same as with the ACLs :

    - eq : match entries whose data is equal to this value
    - ne : match entries whose data is not equal to this value
    - le : match entries whose data is less than or equal to this value
    - ge : match entries whose data is greater than or equal to this value
    - lt : match entries whose data is less than this value
    - gt : match entries whose data is greater than this value

  When the key form is used the entry <key> is removed. The key must be of the same type as the table, which currently is limited to IPv4, IPv6, integer and string.

  Example :
```
        $ echo "show table http_proxy" | socat stdio /tmp/sock1
    >>> # table: http_proxy, type: ip, size:204800, used:2
    >>> 0x80e6a4c: key=127.0.0.1 use=0 exp=3594729 gpc0=0 conn_rate(30000)=1 \
          bytes_out_rate(60000)=187
    >>> 0x80e6a80: key=127.0.0.2 use=0 exp=3594740 gpc0=1 conn_rate(30000)=10 \
          bytes_out_rate(60000)=191

        $ echo "clear table http_proxy key 127.0.0.1" | socat stdio /tmp/sock1

        $ echo "show table http_proxy" | socat stdio /tmp/sock1
    >>> # table: http_proxy, type: ip, size:204800, used:1
    >>> 0x80e6a80: key=127.0.0.2 use=0 exp=3594740 gpc0=1 conn_rate(30000)=10 \
          bytes_out_rate(60000)=191
        $ echo "clear table http_proxy data.gpc0 eq 1" | socat stdio /tmp/sock1
        $ echo "show table http_proxy" | socat stdio /tmp/sock1
    >>> # table: http_proxy, type: ip, size:204800, used:1
```

del acl <acl> [<key>|#<ref>]
  Delete all the acl entries from the acl <acl> corresponding to the key <key>. <acl> is the #<id> or the <file> returned by "show acl". If the <ref> is used, this command delete only the listed reference. The reference can be found with listing the content of the acl. Note that if the reference <acl> is a file and is shared with a map, the entry will be also deleted in the map.

del map <map> [<key>|#<ref>]
  Delete all the map entries from the map <map> corresponding to the key <key>. <map> is the #<id> or the <file> returned by "show map". If the <ref> is used, this command delete only the listed reference. The reference can be found with listing the content of the map. Note that if the reference <map> is a file and is shared with a acl, the entry will be also deleted in the map.

disable agent <backend>/<server>
  Mark the auxiliary agent check as temporarily stopped.

  In the case where an agent check is being run as a auxiliary check, due to the agent-check parameter of a server directive, new checks are only initialised when the agent is in the enabled. Thus, disable agent will prevent any new agent checks from begin initiated until the agent re-enabled using enable agent.

```
13521
13522     When an agent is disabled the processing of an auxiliary agent check that
13523     was initiated while the agent was set as enabled is as follows: All
13524     results that would alter the weight, specifically "drain" or a weight
13525     returned by the agent, are ignored. The processing of agent check is
13526     otherwise unchanged.
13527
13528     The motivation for this feature is to allow the weight changing effects
13529     of the agent checks to be paused to allow the weight of a server to be
13530     configured using set weight without being overridden by the agent.
13531
13532     This command is restricted and can only be issued on sockets configured for
13533     level "admin".
13534
13535   disable frontend <frontend>
13536     Mark the frontend as temporarily stopped. This corresponds to the mode which
13537     is used during a soft restart : the frontend releases the port but can be
13538     enabled again if needed. This should be used with care as some non-Linux OSes
13539     are unable to enable it back. This is intended to be used in environments
13540     where stopping a proxy is not even imaginable but a misconfigured proxy must
13541     be fixed. That way it's possible to release the port and bind it into another
13542     process to restore operations. The frontend will appear with status "STOP"
13543     on the stats page.
13544
13545     The frontend may be specified either by its name or by its numeric ID,
13546     prefixed with a sharp ('#').
13547
13548     This command is restricted and can only be issued on sockets configured for
13549     level "admin".
13550
13551   disable health <backend>/<server>
13552     Mark the primary health check as temporarily stopped. This will disable
13553     sending of health checks, and the last health check result will be ignored.
13554     The server will be in unchecked state and considered UP unless an auxiliary
13555     agent check forces it down.
13556
13557     This command is restricted and can only be issued on sockets configured for
13558     level "admin".
13559
13560   disable server <backend>/<server>
13561     Mark the server DOWN for maintenance. In this mode, no more checks will be
13562     performed on the server until it leaves maintenance.
13563     If the server is tracked by other servers, those servers will be set to DOWN
13564     during the maintenance.
13565
13566     In the statistics page, a server DOWN for maintenance will appear with a
13567     "MAINT" status, its tracking servers with the "MAINT(via)" one.
13568
13569     Both the backend and the server may be specified either by their name or by
13570     their numeric ID, prefixed with a sharp ('#').
13571
13572     This command is restricted and can only be issued on sockets configured for
13573     level "admin".
13574
13575   enable agent <backend>/<server>
13576     Resume auxiliary agent check that was temporarily stopped.
13577
13578     See "disable agent" for details of the effect of temporarily starting
13579     and stopping an auxiliary agent.
13580
13581     This command is restricted and can only be issued on sockets configured for
13582     level "admin".
13583
13584   enable frontend <frontend>
13585     Resume a frontend which was temporarily stopped. It is possible that some of
```



```
13586     the listening ports won't be able to bind anymore (eg: if another process
13587     took them since the 'disable frontend' operation). If this happens, an error
13588     is displayed. Some operating systems might not be able to resume a frontend
13589     which was disabled.
13590
13591     The frontend may be specified either by its name or by its numeric ID,
13592     prefixed with a sharp ('#').
13593
13594     This command is restricted and can only be issued on sockets configured for
13595     level "admin".
13596
13597   enable health <backend>/<server>
13598     Resume a primary health check that was temporarily stopped. This will enable
13599     sending of health checks again. Please see "disable health" for details.
13600
13601     This command is restricted and can only be issued on sockets configured for
13602     level "admin".
13603
13604   enable server <backend>/<server>
13605     If the server was previously marked as DOWN for maintenance, this marks the
13606     server UP and checks are re-enabled.
13607
13608     Both the backend and the server may be specified either by their name or by
13609     their numeric ID, prefixed with a sharp ('#').
13610
13611     This command is restricted and can only be issued on sockets configured for
13612     level "admin".
13613
13614   get map <map> <value>
13615   get acl <acl> <value>
13616     Lookup the value <value> in the map <map> or in the ACL <acl>. <map> or <acl>
13617     are the #<id> or the <file> returned by "show map" or "show acl". This command
13618     returns all the matching patterns associated with this map. This is useful for
13619     debugging maps and ACLs. The output format is composed by one line par
13620     matching type. Each line is composed by space-delimited series of words.
13621
13622     The first two words are:
13623
13624        <match method>:   The match method applied. It can be "found", "bool",
13625                          "int", "ip", "bin", "len", "str", "beg", "sub", "dir",
13626                          "dom", "end" or "reg".
13627
13628        <match result>:   The result. Can be "match" or "no-match".
13629
13630     The following words are returned only if the pattern matches an entry.
13631
13632        <index type>:     "tree" or "list". The internal lookup algorithm.
13633
13634        <case>:           "case-insensitive" or "case-sensitive". The
13635                          interpretation of the case.
13636
13637        <entry matched>:  match="<entry>". Return the matched pattern. It is
13638                          useful with regular expressions.
13639
13640     The two last word are used to show the returned value and its type. With the
13641     "acl" case, the pattern doesn't exist.
13642
13643        return=nothing:        No return because there are no "map".
13644        return="<value>":      The value returned in the string format.
13645        return=cannot-display: The value cannot be converted as string.
13646
13647        type="<type>":         The type of the returned sample.
13648
13649   get weight <backend>/<server>
13650     Report the current weight and the initial weight of server <server> in
```

```
13651     backend <backend> or an error if either doesn't exist. The initial weight is
13652     the one that appears in the configuration file. Both are normally equal
13653     unless the current weight has been changed. Both the backend and the server
13654     may be specified either by their name or by their numeric ID, prefixed with a
13655     sharp ('#').
13656
13657   help
13658     Print the list of known keywords and their basic usage. The same help screen
13659     is also displayed for unknown commands.
13660
13661   prompt
13662     Toggle the prompt at the beginning of the line and enter or leave interactive
13663     mode. In interactive mode, the connection is not closed after a command
13664     completes. Instead, the prompt will appear again, indicating the user that
13665     the interpreter is waiting for a new command. The prompt consists in a right
13666     angle bracket followed by a space "> ". This mode is particularly convenient
13667     when one wants to periodically check information such as stats or errors.
13668     It is also a good idea to enter interactive mode before issuing a "help"
13669     command.
13670
13671   quit
13672     Close the connection when in interactive mode.
13673
13674   set map <map> [<key>|#<ref>] <value>
13675     Modify the value corresponding to each key <key> in a map <map>. <map> is the
13676     #<id> or <file> returned by "show map". If the <ref> is used in place of
13677     <key>, only the entry pointed by <ref> is changed. The new value is <value>.
13678
13679   set maxconn frontend <frontend> <value>
13680     Dynamically change the specified frontend's maxconn setting. Any positive
13681     value is allowed including zero, but setting values larger than the global
13682     maxconn does not make much sense. If the limit is increased and connections
13683     were pending, they will immediately be accepted. If it is lowered to a value
13684     below the current number of connections, new connections acceptation will be
13685     delayed until the threshold is reached. The frontend might be specified by
13686     either its name or its numeric ID prefixed with a sharp ('#').
13687
13688   set maxconn global <maxconn>
13689     Dynamically change the global maxconn setting within the range defined by the
13690     initial global maxconn setting. If it is increased and connections were
13691     pending, they will immediately be accepted. If it is lowered to a value below
13692     the current number of connections, new connections acceptation will be
13693     delayed until the threshold is reached. A value of zero restores the initial
13694     setting.
13695
13696   set rate-limit connections global <value>
13697     Change the process-wide connection rate limit, which is set by the global
13698     'maxconnrate' setting. A value of zero disables the limitation. This limit
13699     applies to all frontends and the change has an immediate effect. The value
13700     is passed in number of connections per second.
13701
13702   set rate-limit http-compression global <value>
13703     Change the maximum input compression rate, which is set by the global
13704     'maxcomprate' setting. A value of zero disables the limitation. The value is
13705     passed in number of kilobytes per second. The value is available in the "show
13706     info" on the line "CompressBpsRateLim" in bytes.
13707
13708   set rate-limit sessions global <value>
13709     Change the process-wide session rate limit, which is set by the global
13710     'maxsessrate' setting. A value of zero disables the limitation. This limit
13711     applies to all frontends and the change has an immediate effect. The value
13712     is passed in number of sessions per second.
13713
13714   set rate-limit ssl-sessions global <value>
13715     Change the process-wide SSL session rate limit, which is set by the global
```



```
13716     'maxsslrate' setting. A value of zero disables the limitation. This limit
13717     applies to all frontends and the change has an immediate effect. The value
13718     is passed in number of sessions per second sent to the SSL stack. It applies
13719     before the handshake in order to protect the stack against handshake abuses.
13720
13721   set server <backend>/<server> agent [ up | down ]
13722     Force a server's agent to a new state. This can be useful to immediately
13723     switch a server's state regardless of some slow agent checks for example.
13724     Note that the change is propagated to tracking servers if any.
13725
13726   set server <backend>/<server> health [ up | stopping | down ]
13727     Force a server's health to a new state. This can be useful to immediately
13728     switch a server's state regardless of some slow health checks for example.
13729     Note that the change is propagated to tracking servers if any.
13730
13731   set server <backend>/<server> state [ ready | drain | maint ]
13732     Force a server's administrative state to a new state. This can be useful to
13733     disable load balancing and/or any traffic to a server. Setting the state to
13734     "ready" puts the server in normal mode, and the command is the equivalent of
13735     the "enable server" command. Setting the state to "maint" disables any traffic
13736     to the server as well as any health checks. This is the equivalent of the
13737     "disable server" command. Setting the mode to "drain" only removes the server
13738     from load balancing but still allows it to be checked and to accept new
13739     persistent connections. Changes are propagated to tracking servers if any.
13740
13741   set server <backend>/<server> weight <weight>[%]
13742     Change a server's weight to the value passed in argument. This is the exact
13743     equivalent of the "set weight" command below.
13744
13745   set ssl ocsp-response <response>
13746     This command is used to update an OCSP Response for a certificate (see "crt"
13747     on "bind" lines). Same controls are performed as during the initial loading of
13748     the response. The <response> must be passed as a base64 encoded string of the
13749     DER encoded response from the OCSP server.
13750
13751     Example:
13752       openssl ocsp -issuer issuer.pem -cert server.pem \
13753                    -host ocsp.issuer.com:80 -respout resp.der
13754       echo "set ssl ocsp-response $(base64 -w 10000 resp.der)" | \
13755                    socat stdio /var/run/haproxy.stat
13756
13757   set table <table> key <key> [data.<data_type> <value>]*
13758     Create or update a stick-table entry in the table. If the key is not present,
13759     an entry is inserted. See stick-table in section 4.2 to find all possible
13760     values for <data_type>. The most likely use consists in dynamically entering
13761     entries for source IP addresses, with a flag in gpc0 to dynamically block an
13762     IP address or affect its quality of service. It is possible to pass multiple
13763     data_types in a single call.
13764
13765   set timeout cli <delay>
13766     Change the CLI interface timeout for current connection. This can be useful
13767     during long debugging sessions where the user needs to constantly inspect
13768     some indicators without being disconnected. The delay is passed in seconds.
13769
13770   set weight <backend>/<server> <weight>[%]
13771     Change a server's weight to the value passed in argument. If the value ends
13772     with the '%' sign, then the new weight will be relative to the initially
13773     configured weight.  Absolute weights are permitted between 0 and 256.
13774     Relative weights must be positive with the resulting absolute weight is
13775     capped at 256.  Servers which are part of a farm running a static
13776     load-balancing algorithm have stricter limitations because the weight
13777     cannot change once set. Thus for these servers, the only accepted values
13778     are 0 and 100% (or 0 and the initial weight). Changes take effect
13779     immediately, though certain LB algorithms require a certain amount of
13780     requests to consider changes. A typical usage of this command is to
```

```
13781     disable a server during an update by setting its weight to zero, then to
13782     enable it again after the update by setting it back to 100%. This command
13783     is restricted and can only be issued on sockets configured for level
13784     "admin". Both the backend and the server may be specified either by their
13785     name or by their numeric ID, prefixed with a sharp ('#').
13786
13787   show errors [<iid>]
13788     Dump last known request and response errors collected by frontends and
13789     backends. If <iid> is specified, the limit the dump to errors concerning
13790     either frontend or backend whose ID is <iid>. This command is restricted
13791     and can only be issued on sockets configured for levels "operator" or
13792     "admin".
13793
13794     The errors which may be collected are the last request and response errors
13795     caused by protocol violations, often due to invalid characters in header
13796     names. The report precisely indicates what exact character violated the
13797     protocol. Other important information such as the exact date the error was
13798     detected, frontend and backend names, the server name (when known), the
13799     internal session ID and the source address which has initiated the session
13800     are reported too.
13801
13802     All characters are returned, and non-printable characters are encoded. The
13803     most common ones (\t = 9, \n = 10, \r = 13 and \e = 27) are encoded as one
13804     letter following a backslash. The backslash itself is encoded as '\\' to
13805     avoid confusion. Other non-printable characters are encoded '\xNN' where
13806     NN is the two-digits hexadecimal representation of the character's ASCII
13807     code.
13808
13809     Lines are prefixed with the position of their first character, starting at 0
13810     for the beginning of the buffer. At most one input line is printed per line,
13811     and large lines will be broken into multiple consecutive output lines so that
13812     the output never goes beyond 79 characters wide. It is easy to detect if a
13813     line was broken, because it will not end with '\n' and the next line's offset
13814     will be followed by a '+' sign, indicating it is a continuation of previous
13815     line.
13816
13817     Example :
13818           $ echo "show errors" | socat stdio /tmp/sock1
13819       >>> [04/Mar/2009:15:46:56.081] backend http-in (#2) : invalid response
13820             src 127.0.0.1, session #54, frontend fe-eth0 (#1), server s2 (#1)
13821             response length 213 bytes, error at position 23:
13822
13823             00000  HTTP/1.0 200 OK\r\n
13824             00017  header/bizarre:blah\r\n
13825             00038  Location: blah\r\n
13826             00054  Long-line: this is a very long line which should b
13827             00104+ e broken into multiple lines on the output buffer,
13828             00154+  otherwise it would be too large to print in a ter
13829             00204+ minal\r\n
13830             00211  \r\n
13831
13832       In the example above, we see that the backend "http-in" which has internal
13833       ID 2 has blocked an invalid response from its server s2 which has internal
13834       ID 1. The request was on session 54 initiated by source 127.0.0.1 and
13835       received by frontend fe-eth0 whose ID is 1. The total response length was
13836       213 bytes when the error was detected, and the error was at byte 23. This
13837       is the slash ('/') in header name "header/bizarre", which is not a valid
13838       HTTP character for a header name.
13839
13840   show info
13841     Dump info about haproxy status on current process.
13842
13843   show map [<map>]
13844     Dump info about map converters. Without argument, the list of all available
13845     maps is returned. If a <map> is specified, its contents are dumped. <map> is
```



```
13846     the #<id> or <file>. The first column is a unique identifier. It can be used
13847     as reference for the operation "del map" and "set map". The second column is
13848     the pattern and the third column is the sample if available. The data returned
13849     are not directly a list of available maps, but are the list of all patterns
13850     composing any map. Many of these patterns can be shared with ACL.
13851
13852   show acl [<acl>]
13853     Dump info about acl converters. Without argument, the list of all available
13854     acls is returned. If a <acl> is specified, its contents are dumped. <acl> if
13855     the #<id> or <file>. The dump format is the same than the map even for the
13856     sample value. The data returned are not a list of available ACL, but are the
13857     list of all patterns composing any ACL. Many of these patterns can be shared
13858     with maps.
13859
13860   show pools
13861     Dump the status of internal memory pools. This is useful to track memory
13862     usage when suspecting a memory leak for example. It does exactly the same
13863     as the SIGQUIT when running in foreground except that it does not flush
13864     the pools.
13865
13866   show sess
13867     Dump all known sessions. Avoid doing this on slow connections as this can
13868     be huge. This command is restricted and can only be issued on sockets
13869     configured for levels "operator" or "admin".
13870
13871   show sess <id>
13872     Display a lot of internal information about the specified session identifier.
13873     This identifier is the first field at the beginning of the lines in the dumps
13874     of "show sess" (it corresponds to the session pointer). Those information are
13875     useless to most users but may be used by haproxy developers to troubleshoot a
13876     complex bug. The output format is intentionally not documented so that it can
13877     freely evolve depending on demands. You may find a description of all fields
13878     returned in src/dumpstats.c
13879
13880     The special id "all" dumps the states of all sessions, which must be avoided
13881     as much as possible as it is highly CPU intensive and can take a lot of time.
13882
13883   show stat [<iid> <type> <sid>]
13884     Dump statistics in the CSV format. By passing <id>, <type> and <sid>, it is
13885     possible to dump only selected items :
13886       - <iid> is a proxy ID, -1 to dump everything
13887       - <type> selects the type of dumpable objects : 1 for frontends, 2 for
13888         backends, 4 for servers, -1 for everything. These values can be ORed,
13889         for example:
13890            1 + 2     = 3   -> frontend + backend.
13891            1 + 2 + 4 = 7   -> frontend + backend + server.
13892       - <sid> is a server ID, -1 to dump everything from the selected proxy.
13893
13894     Example :
13895           $ echo "show info;show stat" | socat stdio unix-connect:/tmp/sock1
13896       >>> Name: HAProxy
13897           Version: 1.4-dev2-49
13898           Release_date: 2009/09/23
13899           Nbproc: 1
13900           Process_num: 1
13901           (...)
13902
13903           # pxname,svname,qcur,qmax,scur,smax,slim,stot,bin,bout,dreq,  (...)
13904           stats,FRONTEND,,,0,0,1000,0,0,0,0,0,,,,,OPEN,,,,,,,,,1,1,0, (...)
13905           stats,BACKEND,0,0,0,0,1000,0,0,0,0,0,,0,0,0,0,UP,0,0,0,,0,250,(...)
13906           (...)
13907           www1,BACKEND,0,0,0,0,1000,0,0,0,0,0,,0,0,0,0,UP,1,1,0,,0,250, (...)
13908
13909           $
13910
```

```
13911      Here, two commands have been issued at once. That way it's easy to find
13912      which process the stats apply to in multi-process mode. Notice the empty
13913      line after the information output which marks the end of the first block.
13914      A similar empty line appears at the end of the second block (stats) so that
13915      the reader knows the output has not been truncated.
13916
13917  show table
13918    Dump general information on all known stick-tables. Their name is returned
13919    (the name of the proxy which holds them), their type (currently zero, always
13920    IP), their size in maximum possible number of entries, and the number of
13921    entries currently in use.
13922
13923    Example :
13924          $ echo "show table" | socat stdio /tmp/sock1
13925        >>> # table: front_pub, type: ip, size:204800, used:171454
13926        >>> # table: back_rdp, type: ip, size:204800, used:0
13927
13928  show table <name> [ data.<type> <operator> <value> ] | [ key <key> ]
13929    Dump contents of stick-table <name>. In this mode, a first line of generic
13930    information about the table is reported as with "show table", then all
13931    entries are dumped. Since this can be quite heavy, it is possible to specify
13932    a filter in order to specify what entries to display.
13933
13934    When the "data." form is used the filter applies to the stored data (see
13935    "stick-table" in section 4.2).  A stored data type must be specified
13936    in <type>, and this data type must be stored in the table otherwise an
13937    error is reported. The data is compared according to <operator> with the
13938    64-bit integer <value>.  Operators are the same as with the ACLs :
13939
13940      - eq : match entries whose data is equal to this value
13941      - ne : match entries whose data is not equal to this value
13942      - le : match entries whose data is less than or equal to this value
13943      - ge : match entries whose data is greater than or equal to this value
13944      - lt : match entries whose data is less than this value
13945      - gt : match entries whose data is greater than this value
13946
13947
13948    When the key form is used the entry <key> is shown.  The key must be of the
13949    same type as the table, which currently is limited to IPv4, IPv6, integer,
13950    and string.
13951
13952    Example :
13953          $ echo "show table http_proxy" | socat stdio /tmp/sock1
13954        >>> # table: http_proxy, type: ip, size:204800, used:2
13955        >>> 0x80e6a4c: key=127.0.0.1 use=0 exp=3594729 gpc0=0 conn_rate(30000)=1  \
13956              bytes_out_rate(60000)=187
13957        >>> 0x80e6a80: key=127.0.0.2 use=0 exp=3594740 gpc0=1 conn_rate(30000)=10 \
13958              bytes_out_rate(60000)=191
13959
13960          $ echo "show table http_proxy data.gpc0 gt 0" | socat stdio /tmp/sock1
13961        >>> # table: http_proxy, type: ip, size:204800, used:2
13962        >>> 0x80e6a80: key=127.0.0.2 use=0 exp=3594740 gpc0=1 conn_rate(30000)=10 \
13963              bytes_out_rate(60000)=191
13964
13965          $ echo "show table http_proxy data.conn_rate gt 5" | \
13966              socat stdio /tmp/sock1
13967        >>> # table: http_proxy, type: ip, size:204800, used:2
13968        >>> 0x80e6a80: key=127.0.0.2 use=0 exp=3594740 gpc0=1 conn_rate(30000)=10 \
13969              bytes_out_rate(60000)=191
13970
13971          $ echo "show table http_proxy key 127.0.0.2" | \
13972              socat stdio /tmp/sock1
13973        >>> # table: http_proxy, type: ip, size:204800, used:2
13974        >>> 0x80e6a80: key=127.0.0.2 use=0 exp=3594740 gpc0=1 conn_rate(30000)=10 \
13975              bytes_out_rate(60000)=191
```



```
13976
13977    When the data criterion applies to a dynamic value dependent on time such as
13978    a bytes rate, the value is dynamically computed during the evaluation of the
13979    entry in order to decide whether it has to be dumped or not. This means that
13980    such a filter could match for some time then not match anymore because as
13981    time goes, the average event rate drops.
13982
13983    It is possible to use this to extract lists of IP addresses abusing the
13984    service, in order to monitor them or even blacklist them in a firewall.
13985    Example :
13986          $ echo "show table http_proxy data.gpc0 gt 0" \
13987            | socat stdio /tmp/sock1 \
13988            | fgrep 'key=' | cut -d' ' -f2 | cut -d= -f2 > abusers-ip.txt
13989            ( or | awk '/key/{ print a[split($2,a,"=")]; }' )
13990
13991  shutdown frontend <frontend>
13992    Completely delete the specified frontend. All the ports it was bound to will
13993    be released. It will not be possible to enable the frontend anymore after
13994    this operation. This is intended to be used in environments where stopping a
13995    proxy is not even imaginable but a misconfigured proxy must be fixed. That
13996    way it's possible to release the port and bind it into another process to
13997    restore operations. The frontend will not appear at all on the stats page
13998    once it is terminated.
13999
14000    The frontend may be specified either by its name or by its numeric ID,
14001    prefixed with a sharp ('#').
14002
14003    This command is restricted and can only be issued on sockets configured for
14004    level "admin".
14005
14006  shutdown session <id>
14007    Immediately terminate the session matching the specified session identifier.
14008    This identifier is the first field at the beginning of the lines in the dumps
14009    of "show sess" (it corresponds to the session pointer). This can be used to
14010    terminate a long-running session without waiting for a timeout or when an
14011    endless transfer is ongoing. Such terminated sessions are reported with a 'K'
14012    flag in the logs.
14013
14014  shutdown sessions server <backend>/<server>
14015    Immediately terminate all the sessions attached to the specified server. This
14016    can be used to terminate long-running sessions after a server is put into
14017    maintenance mode, for instance. Such terminated sessions are reported with a
14018    'K' flag in the logs.
14019
14020  /*
14021   * Local variables:
14022   *  fill-column: 79
14023   * End:
14024   */
```