VCL (/vcl)

Cryptographic (/vcl/cryptographic/)

Notes

In base64 decoding, the output theoretically could be in binary but is interpreted as a string. So if the binary output contains '\0' then it could be truncated.

The time based One-Time Password algorithm initializes the HMAC using the key and appropriate hash type. Then it hashes the message

`(<time now in seconds since UNIX epoch> / <interval>) + <offset>`

as a 64bit unsigned integer (little endian) and base64 encodes the result.

Examples

One-Time Password Validation (Token Authentication)

Use this to validate tokens with a URL format like the following:

`http://cname-to-fastly/video.mp4?6h2YUl1CB4C50SbkZ0E6U3dZGjh+84dz3+Zope2Uhik=`

Example implementations for token generation in various languages can be found in GitHub (https://github.com/fastly/token-functions).

Example VCL

```
sub vcl_recv {

  /* make sure there is a token */
  if (req.url !~ "[?&]token=([^&]+)") {
    error 403;
  }

  if (re.group.1 != digest.time_hmac_sha256("RmFzdGx5IFRva2VuIFRlc3Q=", 60, 0) &&
      re.group.1 != digest.time_hmac_sha256("RmFzdGx5IFRva2VuIFRlc3Q=", 60, -1)) {
    error 403;
  }

#FASTLY recv

  ...
}
```

Signature

```
set resp.http.x-data-sig = digest.hmac_sha256("secretkey",resp.http.x-data);
```

Base64 decoding

A snippet like this in `vcl_error` would set the response body to the value of the request header field named `x-parrot` after base64-decoding the value:

```
synthetic digest.base64_decode(req.http.x-parrot);
```

However, if the base64-decoded string contains a NUL byte (0x00), then that byte and any bytes following it will not be included in the response. Keep that in mind if you intend to send a synthetic response that contains binary data. There is currently no way to send a synthetic response containing a NUL byte.

Cryptographic Functions

digest.awsv4_hmac (/vcl/functions/digest-awsv4-hmac/)

Returns an AWSv4 message authentication code (https://docs.aws.amazon.com/AmazonS3/latest/API/sig-v4-authenticating-requests.html#signing-request-intro) based on the supplied `key` and `string`. This function automatically prepends "AWS4" in front of the secret access key (the first function parameter) as required by the protocol. This function does not support binary data for its `key` or `string` parameters.

Format

```
STRING (/vcl/types/string)
awsv4_hmac(STRING key, STRING date_stamp, STRING region, STRING service, STRING string)
```

Examples

```
set resp.http.sig = digest.awsv4_hmac(
    "wJalrXUtnFEMI/K7MDENG+bPxRfiCYEXAMPLEKEY",
    "20120215",
    "us-east-1",
    "iam",
    "hello");
```

digest.base64_decode (/vcl/functions/digest-base64-decode/)

Decode Base64. Returns a string.

Format

```
digest.base64_decode(string)
```

digest.base64 (/vcl/functions/digest-base64/)

Base64 (https://en.wikipedia.org/wiki/Base64) encoding. Returns the base64-encoded version of the input-string.

Format

```
digest.base64(string)
```

digest.base64url_decode (/vcl/functions/digest-base64url-decode/)

Decode Base64 with url safe characters in. Returns a string.

Format

```
digest.base64url_decode(string)
```

digest.base64url_nopad_decode (/vcl/functions/digest-base64url-nopad-decode/)

Decode Base64 with url safe characters. Returns a string. Identical to base64_url_decode.

Format

```
digest.base64url_decode(string)
```

digest.base64url_nopad (/vcl/functions/digest-base64url-nopad/)

Base64 (https://en.wikipedia.org/wiki/Base64#URL_applications) encoding. Returns the base64-encoded version of the input-string. Replaces +/ with -_ for url safety. Has no length padding (https://en.wikipedia.org/wiki/Base64#Padding).

Format

```
digest.base64url_nopad(string)
```

digest.base64url (/vcl/functions/digest-base64url/)

Base64 (https://en.wikipedia.org/wiki/Base64#URL_applications) encoding. Returns the base64-encoded version of the input-string. Replaces +/ with -_ for url safety.

Format

```
digest.base64url(string)
```

digest.hash_crc32 (/vcl/functions/digest-hash-crc32/)

Use a 32 bit Cyclic Redundancy Checksum (https://en.wikipedia.org/wiki/CRC32). Returns a hex-encoded string.

Format

```
digest.hash_crc32(string)
```

digest.hash_crc32b (/vcl/functions/digest-hash-crc32b/)

A reversed CRC32 (for compatibility with some PHP applications (http://php.net/manual/en/function.hash-file.php#104836)). Returns a hex-encoded string.

Format

```
digest.hash_crc32b(string)
```

digest.hash_md5 (/vcl/functions/digest-hash-md5/)

Use the MD5 (https://en.wikipedia.org/wiki/MD5) hash. Returns a hex-encoded string.

Format

```
digest.hash_md5(string)
```

digest.hash_sha1 (/vcl/functions/digest-hash-sha1/)

Use the SHA1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

Format

```
digest.hash_sha1(string)
```

digest.hash_sha224 (/vcl/functions/digest-hash-sha224/)

Use the SHA224 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

Format

```
digest.hash_sha224(string)
```

digest.hash_sha256 (/vcl/functions/digest-hash-sha256/)

Use the SHA256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

Format

```
digest.hash_sha256(string)
```

digest.hash_sha384 (/vcl/functions/digest-hash-sha384/)

Use the SHA384 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

Format

```
digest.hash_sha384(string)
```

digest.hash_sha512 (/vcl/functions/digest-hash-sha512/)

Use the SHA512 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

Format

```
digest.hash_sha512(string)
```

digest.hmac_md5_base64 (/vcl/functions/digest-hmac-md5-base64/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using MD5. Returns a base64-encoded (https://en.wikipedia.org/wiki/Base64) string.

Format

```
digest.hmac_md5_base64(key, message)
```

digest.hmac_md5 (/vcl/functions/digest-hmac-md5/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using MD5. Returns a hex-encoded string prepended with 0x.

Format

```
digest.hmac_md5(key, message)
```

digest.hmac_sha1_base64 (/vcl/functions/digest-hmac-sha1-base64/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a base64-encoded (https://en.wikipedia.org/wiki/Base64) string.

Format

```
digest.hmac_sha1_base64(key, message)
```

digest.hmac_sha1 (/vcl/functions/digest-hmac-sha1/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a hex-encoded string prepended with 0x.

Format

```
digest.hmac_sha1(key, message)
```

digest.hmac_sha256_base64 (/vcl/functions/digest-hmac-sha256-base64/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a base64-encoded (https://en.wikipedia.org/wiki/Base64) string.

Format

```
digest.hmac_sha256_base64(key, message)
```

digest.hmac_sha256 (/vcl/functions/digest-hmac-sha256/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a hex-encoded string prepended with 0x.

Format

```
digest.hmac_sha256(key, message)
```

digest.rsa_verify (/vcl/functions/digest-rsa-verify/)

A boolean function that returns true if the RSA signature of `payload` using `public_key` matches `digest`. The `hash_method` parameter selects the digest function to use. It can be `sha256`, `sha384`, `sha512`, or `default` (`default` is equivalent to `sha256`). The `STRING_LIST` parameter in the payload/digest could reference headers such as `req.http.payload` and `req.http.digest`. The `base64_method` parameter is optional. It can be `standard`, `url`, `url_nopad`, or `default` (`default` is equivalent to `url_nopad`).

Format

```
digest.rsa_verify(ID hash_method, STRING_LIST public_key, STRING_LIST payload, STRING_LIST dige
st [, ID base64_method ])
```

Examples

```
if (digest.rsa_verify(sha256, {"-----BEGIN PUBLIC KEY-----
aabbccddIieEffggHHhEXAMPLEPUBLICKEY
-----END PUBLIC KEY-----"}, req.http.payload, req.http.digest, url_nopad)) {
  set req.http.verified = "Verified";
} else {
  set req.http.verified = "Not Verified";
}
error 900;
```

digest.secure_is_equal (/vcl/functions/digest-secure-is-equal/)

A boolean function that returns true if s1 and s2 are equal. The comparison is done in constant time to defend against timing attacks.

Format

```
digest.secure_is_equal(STRING_LIST s1, STRING_LIST s2)
```

digest.time_hmac_md5 (/vcl/functions/digest-time-hmac-md5/)

Time based One Time Password using MD5. Returns base64 encoded output.

Format

```
digest.time_hmac_md5(base64 encoded key, interval, offset)
```

digest.time_hmac_sha1 (/vcl/functions/digest-time-hmac-sha1/)

Time based One Time Password using SHA1. Returns base64 encoded output.

Format

```
digest.time_hmac_sha1(base64 encoded key, interval, offset)
```

digest.time_hmac_sha256 (/vcl/functions/digest-time-hmac-sha256/)

Time based One Time Password using SHA256. Returns base64 encoded output.

Format

```
digest.time_hmac_sha256(base64 encoded key, interval, offset)
```

Date and time (/vcl/date-and-time/)

Date and time Functions

std.integer2time (/vcl/functions/std-integer2time/)

Converts an integer to a time variable. To use a string, use `std.atoi`.

Comparison operators like `> < >= <= == !=` do not work with `std.time` or `std.integer2time`. Instead, you can compare two times using something similar to this:

```
if (time.is_after(now, std.integer2time(std.atoi("1445445162")))) {
  # do something
}
```

Format

```
std.integer2time(seconds_since_epoch)
```

Examples

```
std.integer2time(std.atoi("1445445162"));
```

std.time (/vcl/functions/std-time/)

Converts a string to a time variable. The following string formats are supported: `Sun, 06 Nov 1994 08:49:37 GMT`, `Sunday, 06-Nov-94 08:49:37 GMT`, `Sun Nov  6 08:49:37 1994`, `784111777.00`, `784111777`. Useful because time variables are needed as arguments for functions like `time.add` and `strftime`.

Comparison operators like `> < >= <= == !=` do not work with `std.time` or `std.integer2time`. Instead, you can compare two times using something similar to this:

```
if (time.is_after(now, std.integer2time(std.atoi("1445445162")))) {
  # do something
}
```

Format

```
std.time(string_to_parse, fallback_value)
```

Examples

```
set resp.http.X-Seconds-Since-Modified = strftime({"%s"}, time.sub(now, std.time(resp.http.Last
-Modified, now)));
```

strftime (/vcl/functions/strftime/)

Formats a time to a string. This uses standard POSIX strftime formats (https://www.unix.com/man-page/FreeBSD/3/strftime/).

★ **TIP:** Regular strings ("short strings") in VCL use `%xx` escapes (percent encoding) for special characters, which would conflict with the `%` used in the strftime format. For the strftime examples, we use VCL "long strings" `{"..."}`, which do not use the `%xx` escapes. Alternatively, you could use `%25` for each `%`.

Format

```
strftime(format, time)
```

Examples

```
set resp.http.Now = strftime({"%Y-%m-%d %H:%M"}, now)
```

```
set resp.http.Start = strftime({"%a, %d %b %Y %T %z"}, time.start)
```

time.add (/vcl/functions/time-add/)

Adds `offset` to `time`.

Format

```
time.add(time, offset)
```

Examples

```
if (time.is_after(time.add(now, 10m), time.hex_to_time(1, "d0542d8"))) {
  ...
}
```

time.hex_to_time (/vcl/functions/time-hex-to-time/)

Takes a hexadecimal string value, divides by `divider` and interprets the result as seconds since UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).

Format

```
time.hex_to_time(divider, hextime)
```

Examples

```
if (time.is_after(time.add(now, 10m), time.hex_to_time(1, "d0542d8"))) {
  ...
}
```

time.is_after (/vcl/functions/time-is-after/)

Returns `TRUE` if `time1` is after `time2`. (Normal timeflow and causality required.)

Format

```
time.is_after(time1, time2)
```

Examples

```
if (time.is_after(time.add(now, 10m), time.hex_to_time(1, "d0542d8"))) {
  ...
}
```

time.sub (/vcl/functions/time-sub/)

Subtracts `offset` from `time`.

Format

```
time.sub(time, offset)
```

Date and time Variables

now.sec (/vcl/variables/now-sec/)

Like the `now` variable, but in seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).

Readable From

All subroutines

now (/vcl/variables/now/)

The current time in RFC 1123 format (https://www.ietf.org/rfc/rfc1123.txt) format.

Readable From

All subroutines

time.elapsed.msec_frac (/vcl/variables/time-elapsed-msec-frac/)

The time the request started in milliseconds since the last whole second.

Readable From

- `vcl_deliver`
- `vcl_log`

time.elapsed.msec (/vcl/variables/time-elapsed-msec/)

The time since the request start in milliseconds.

Readable From

- `vcl_deliver`
- `vcl_log`

time.elapsed.sec (/vcl/variables/time-elapsed-sec/)

The time since the request start in seconds.

Readable From

- `vcl_deliver`
- `vcl_log`

time.elapsed.usec_frac (/vcl/variables/time-elapsed-usec-frac/)

The time the request started in microseconds since the last whole second.

Readable From

- `vcl_deliver`
- `vcl_log`

time.elapsed.usec (/vcl/variables/time-elapsed-usec/)

The time since the request start in microseconds.

Readable From

- `vcl_deliver`
- `vcl_log`

time.elapsed (/vcl/variables/time-elapsed/)

The time since the request start, using RFC 1123 format (https://www.ietf.org/rfc/rfc1123.txt). Also useful for strftime.

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.msec_frac (/vcl/variables/time-end-msec-frac/)

The time the request started in milliseconds since the last whole second.

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.msec (/vcl/variables/time-end-msec/)

The time the request ended in milliseconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.sec (/vcl/variables/time-end-sec/)

The time the request ended in seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.usec_frac (/vcl/variables/time-end-usec-frac/)

The time the request started in microseconds since the last whole second.

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.usec (/vcl/variables/time-end-usec/)

The time the request ended in microseconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time).

Readable From

- `vcl_deliver`
- `vcl_log`

time.end (/vcl/variables/time-end/)

The time the request ended, using RFC 1123 format (https://www.ietf.org/rfc/rfc1123.txt). Also useful for strftime.

Readable From

- `vcl_deliver`
- `vcl_log`

time.start.msec_frac (/vcl/variables/time-start-msec-frac/)

The time the request started in milliseconds since the last whole second, after TLS termination.

Readable From

All subroutines

time.start.msec (/vcl/variables/time-start-msec/)

The time the request started in milliseconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time), after TLS termination.

Readable From

All subroutines

time.start.sec (/vcl/variables/time-start-sec/)

The time the request started in seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time), after TLS termination.

Readable From

All subroutines

time.start.usec_frac (/vcl/variables/time-start-usec-frac/)

The time the request started in microseconds since the last whole second, after TLS termination.

Readable From

All subroutines

time.start.usec (/vcl/variables/time-start-usec/)

The time the request started in microseconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time), after TLS termination.

Readable From

All subroutines

time.start (/vcl/variables/time-start/)

The time the request started, after TLS termination, using RFC 1123 format (https://www.ietf.org/rfc/rfc1123.txt).

Readable From

All subroutines

time.to_first_byte (/vcl/variables/time-to-first-byte/)

The time interval since the request started up to the point before the `vcl_deliver` function ran. When used in a string context, an RTIME variable like this one will be formatted as a number in seconds with 3 decimal digits of precision. In `vcl_deliver` this interval will be very close to `time.elapsed`. In `vcl_log`, the difference between `time.elapsed` and `time.to_first_byte` will be the time that it took to send the response body.

Readable From

- `vcl_deliver`
- `vcl_log`

Geolocation (/vcl/geolocation/)

NOTE: While Fastly exposes these geographic variables, we cannot guarantee their accuracy. The variables are based on available geographic data and are intended to provide an approximate location of where requests might be coming from, rather than an exact location. The postal code associated with an IP address is the most granular level of geographic data available.

NOTE: Geolocation information, including data streamed by our log streaming service (/guides/streaming-logs/), is intended to be used only in connection with your use of Fastly services. Use of geolocation data for other purposes may require the permission of a IP geolocation dataset vendor, such as Digital Element (https://www.digitalelement.com/end-user-license-agreement-eula/).

TIP: If you're updating your configurations from older version of the geolocation variables, be sure to read our migration guide (/guides/migrations/migrating-geolocation-variables-to-the-new-dataset).

Using geographic variables with shielding

If you have shielding (/guides/performance-tuning/shielding) enabled, you should set the following variable before using geographic variables:

```
set client.geo.ip_override = req.http.Fastly-Client-IP;
```

Geolocation Variables

client.as.name (/vcl/variables/client-as-name/)

The name of the organization associated with `client.as.number`.

Readable From

All subroutines

client.as.number (/vcl/variables/client-as-number/)

The autonomous system (AS) (https://en.wikipedia.org/wiki/Autonomous_system_(Internet)) number associated with this IP address.

Readable From

All subroutines

client.geo.area_code (/vcl/variables/client-geo-area-code/)

The telephone area code associated with the IP address. These are only available for IP addresses in the United States.

Readable From

All subroutines

client.geo.city.ascii (/vcl/variables/client-geo-city-ascii/)

An alias of `client.geo.city`.

Readable From

All subroutines

client.geo.city.utf8 (/vcl/variables/client-geo-city-utf8/)

The city or town name associated with the IP address, encoded using the UTF-8 character encoding.

Readable From

All subroutines

client.geo.city (/vcl/variables/client-geo-city/)

The city or town name associated with the IP address, encoded using the ASCII character encoding (a lowercase ASCII approximation of the original string with diacritics removed).

Readable From

All subroutines

client.geo.conn_speed (/vcl/variables/client-geo-conn-speed/)

The type of connection speed (https://www.webopedia.com/quick_ref/internet_connection_types.asp) associated with the IP address. Possible values are: **broadband**, **cable**, **dialup**, **mobile**, **oc12**, **oc3**, **t1**, **t3**, **satellite**, **wireless**, **xdsl**.

Readable From

All subroutines

client.geo.continent_code (/vcl/variables/client-geo-continent-code/)

A two-character code representing the continent associated with the IP address. Possible codes are: **AF** - Africa, **AS** - Asia, **EU** - Europe, **NA** - North America, **OC** - Oceania, **SA** - South America, **AN** - Antarctica.

Readable From

All subroutines

client.geo.country_code (/vcl/variables/client-geo-country-code/)

A two-character ISO 3166-1 (https://en.wikipedia.org/wiki/ISO_3166-1) country code for the country associated with the IP address. The **US** country code is returned for IP addresses associated with overseas United States military bases.

Readable From

All subroutines

client.geo.country_code3 (/vcl/variables/client-geo-country-code3/)

A three-character ISO 3166-1 alpha-3 (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-3) country code for the country associated with the IP address. The **USA** country code is returned for IP addresses associated with overseas United States military bases.

Readable From

All subroutines

client.geo.country_name.ascii (/vcl/variables/client-geo-country-name-ascii/)

An alias of `client.geo.country_name`.

Readable From

All subroutines

client.geo.country_name.ascii (/vcl/variables/client-geo-country-name-utf8/)

The country name associated with the IP address, encoded using the UTF-8 character encoding.

Readable From

All subroutines

client.geo.country_name (/vcl/variables/client-geo-country-name/)

The country name associated with the IP address, encoded using the ASCII character encoding (a lowercase ASCII approximation of the original string with diacritics removed).

Readable From

All subroutines

client.geo.gmt_offset (/vcl/variables/client-geo-gmt-offset/)

The time zone offset from coordinated universal time (UTC) for the `client.geo.city` associated with the IP address.

Readable From

All subroutines

client.geo.latitude (/vcl/variables/client-geo-latitude/)

The latitude associated with the IP address.

Readable From

All subroutines

client.geo.longitude (/vcl/variables/client-geo-longitude/)

The longitude associated with the IP address.

Readable From

All subroutines

client.geo.metro_code (/vcl/variables/client-geo-metro-code/)

The metro code associated with the IP address. Metro codes represent designated market areas (DMAs) in the United States and Germany, Independent Television Service (ITV) regions in the UK, department codes in France, South Korean administrative divisions (si, gun, gu or cities, counties, and districts), Chinese administrative regions (diji shi, or "region-level" cities), Russian federal districts, Norwegian municipalities, urban areas in New Zealand, and the Greater Capital City Statistical Area (GCCSA) and Significant Urban Areas (SUAs) in Australia.

Readable From

All subroutines

client.geo.postal_code (/vcl/variables/client-geo-postal-code/)

The postal code associated with the IP address. These are available for some IP addresses in Australia, Canada, France, Germany, Italy, Spain, Switzerland, the United Kingdom, and the United States. We return the first 3 characters for Canadian postal codes. We return the first 2-4 characters (outward code) for postal codes in the United Kingdom.

Readable From

All subroutines

client.geo.region (/vcl/variables/client-geo-region/)

The ISO 3166-2 (https://en.wikipedia.org/wiki/ISO_3166-2) region code associated with the IP address.

Readable From

All subroutines

server.datacenter (/vcl/variables/server-datacenter/)

A code representing one of Fastly's POP locations (/guides/basic-concepts/fastly-pop-locations).

Readable From

All subroutines

server.region (/vcl/variables/server-region/)

A code representing the general region of the world in which the POP location resides. One of the following:

Region Name	Approximate Geographic Location of Fastly POPs
APAC	Australia and New Zealand
Asia	throughout the Asian continent (except India)
Asia-South	southern Asia
EU	the European continent
North-America	Canada
SA-East	eastern South America
SA-South	southern South America

Region Name	Approximate Geographic Location of Fastly POPs
South-Africa	the southern regions of Africa
US-Central	the central United States
US-East	the eastern United States
US-West	the western United States

Readable From

All subroutines

Miscellaneous (/vcl/miscellaneous/)

Miscellaneous features

Feature	Description
`goto`	Performs a one-way transfer of control to another line of code. See the example for more information.
`return`	Returns (with no return value) from a custom subroutine to exit early. See the example for more information.

Examples

Use the following examples to learn how to implement the features.

Goto

Similar to some programming languages, `goto` statements in VCL allow you perform a one-way transfer of control to another line of code. When using `goto`, jumps must always be forward, rather than to an earlier part of code.

This act of "jumping" allows you to do things like perform logical operations or set headers before returning lookup, error, or pass actions. These statements also make it easier to do things like jump to common error handling blocks before returning from a function. The `goto` statement works in custom subroutines.

```
sub vcl_recv {
  if (!req.http.Foo) {
    goto foo;
  }

foo:
  set req.http.Foo = "1";
}
```

Return

You can use `return` to exit early from a custom subroutine.

```
sub custom_subroutine {
  if (req.http.Cookie:user_id) {
    return;
  }

  # do a bunch of other stuff
}
```

Miscellaneous Functions

cstr_escape (/vcl/functions/cstr-escape/)

Escapes characters unsafe for printing from a string. For example, `"` becomes `\"`.

Format

```
STRING (/vcl/types/string)
cstr_escape(STRING cstr)
```

Examples

```
log "escaped=" cstr_escape("%22" req.protocol "%22")
```

http_status_matches (/vcl/functions/http-status-matches/)

Determines whether or not an HTTP status code matches a pattern. The arguments are an integer (usually `beresp.status` or `resp.status`) and a comma-separated list of status codes, optionally prefixed by a `!` to negate the match. It returns `TRUE` or `FALSE`.

Format

```
BOOL (/vcl/types/bool)
http_status_matches(INT status, STRING fmt)
```

Examples

```
if (http_status_matches(beresp.status, "!200,304")) {
  restart;
}
```

if() (/vcl/functions/if/)

Implements a ternary operator for strings; if the expression is true, it returns `TRUE`; if the expression is false, it returns `FALSE`. For example, you have an `if(x, true-expression, false-expression);` if this argument is true, the `true-expression` is returned. Otherwise, the `false-expression` is returned.

You can use `if()` as a construct to make simple conditional expressions more concise.

Format

```
STRING (/vcl/types/string)
if(BOOL expression, STRING valueiftrue, STRING valueiffalse)
```

Examples

```
set req.http.foo-status = if(req.http.foo, "present", "absent");
```

std.atoi (/vcl/functions/std-atoi/)

Takes a string (which represents an integer) as an argument and returns its value.

Format

```
INTEGER (/vcl/types/integer)
std.atoi(STRING s)
```

Examples

```
if (std.atoi(req.http.X-Decimal) == 42) {
  set req.http.X-TheAnswer = "Found";
}
```

std.ip (/vcl/functions/std-ip/)

An alias of `std.str2ip`.

Format

```
IP (/vcl/types/ip)
std.ip(STRING addr, STRING fallback)
```

std.ip2str (/vcl/functions/std-ip2str/)

Converts the IP address (v4 or v6) to a string.

Format

```
STRING (/vcl/types/string)
std.ip2str(IP ip)
```

Examples

```
if (std.ip2str(std.str2ip("192.0.2.1", "192.0.2.2")) == "192.0.2.1") {
```

std.str2ip (/vcl/functions/std-str2ip/)

Converts the string address to an IP address (v4 or v6). For example, `if (std.str2ip("192.0.2.1", "192.0.2.2") ~ my_acl) {` where `192.0.2.2` is the fallback. If conversion fails, the fallback will be returned. Note that only the first result from DNS resolution is returned.

Format

```
IP (/vcl/types/ip)
std.str2ip(STRING addr, STRING fallback)
```

std.strlen (/vcl/functions/std-strlen/)

Returns the length of the string. For example, `std.strlen("Hello world!");` will return `12` (because the string includes whitespaces and punctuation).

Format

```
INTEGER (/vcl/types/integer)
std.strlen(STRING s)
```

Examples

```
if (std.strlen(req.http.Cookie) > 1024) {
  unset req.http.Cookie;
}
```

std.strstr (/vcl/functions/std-strstr/)

Finds the first occurrence of a byte string and returns its value.

Format

```
STRING (/vcl/types/string)
std.strstr(STRING haystack, STRING needle)
```

Examples

```
set req.http.X-qs = std.strstr(req.url, "?");
```

std.strtol (/vcl/functions/std-strtol/)

Converts a string to an integer, using the second argument as base. Base can be `2` to `36`, or `0`. A `0` base means that base 10 (decimal) is used, unless the string has a `0x` or `0` prefix, in which case base 16 (hexadecimal) and base 8 (octal) are used respectively. For example, `std.strtol("0xa0", 0)` will return `160`.

Format

```
INTEGER (/vcl/types/integer)
std.strtol(STRING s, UINT base)
```

Examples

```
if (std.strtol(req.http.X-HexValue, 16) == 42) {
  set req.http.X-TheAnswer = "Found";
}
```

std.tolower (/vcl/functions/std-tolower/)

Changes the case of a string to lower case. For example, `std.tolower("HELLO");` will return `"hello"`.

Format

```
STRING (/vcl/types/string)
std.tolower(STRING_LIST s)
```

Examples

```
set beresp.http.x-nice = std.tolower("VerY");
```

std.toupper (/vcl/functions/std-toupper/)

Changes the case of a string to upper case. For example, `std.toupper("hello");` will return `"HELLO"`.

Format

```
STRING (/vcl/types/string)
std.toupper(STRING_LIST s)
```

Examples

```
set beresp.http.x-scream = std.toupper("yes!");
```

subfield (/vcl/functions/subfield/)

Provides a means to access subfields from a header like `Cache-Control`, `Cookie`, and `Edge-Control`.

The optional separator character parameter defaults to `,`. It can be any one-character constant. For example, `;` is a useful separator for extracting parameters from a Set-Cookie field.

This functionality is also achievable by using the `:` accessor within a variable name. When the subfield is a valueless token (like "private" in the case of `Cache-Control: max-age=1200, private`), an empty string is returned.

The `:` accessor also works for retrieving variables in a cookie.

This function is not prefixed with the `std.` namespace.

Format

```
STRING subfield(STRING header, STRING fieldname [, STRING separator_character])
```

Examples

```
if (subfield(beresp.http.Cache-Control, "private")) {
  return (pass);
}

set beresp.ttl = beresp.http.Cache-Control:max-age;
set beresp.http.Cache-Control:max-age = "1200";

if (subfield(beresp.http.Set-Cookie, "httponly", ";")) {
  ....
}
```

urldecode (/vcl/functions/urldecode/)

Decodes a percent-encoded string. For example, `urldecode({"hello%20world+!"});` and `urldecode("hello%2520world+!");` will both return `"hello world !"`

Format

```
STRING (/vcl/types/string)
urldecode(STRING input)
```

Examples

```
set req.http.X-Cookie = regsub(req.url, ".*\?cookie=", ""); set req.http.Cookie = urldecode(req
.http.X-Cookie);
```

urlencode (/vcl/functions/urlencode/)

Encodes a string for use in a URL. This is also known as percent-encoding (https://en.wikipedia.org/wiki/Percent-encoding). For example, `urlencode("hello world");` will return `"hello%20world"`.

Format

```
STRING (/vcl/types/string)
urlencode(STRING input)
```

Examples

```
set req.url = req.url "?cookie=" urlencode(req.http.Cookie);
```

Miscellaneous Variables

bereq.url.basename (/vcl/variables/bereq-url-basename/)

Same as `req.url.basename`, except for use between Fastly and your origin servers.

bereq.url.dirname (/vcl/variables/bereq-url-dirname/)

Same as `req.url.dirname`, except for use between Fastly and your origin servers.

bereq.url.qs (/vcl/variables/bereq-url-qs/)

The query string portion of `bereq.url`. This will be from immediately after the `?` to the end of the URL.

bereq.url (/vcl/variables/bereq-url/)

The URL sent to the backend. Does not include the host and scheme, meaning in `www.example.com/index.html`, `bereq.url` would contain `/index.html`.

beresp.backend.ip (/vcl/variables/beresp-backend-ip/)

The IP of the backend this response was fetched from (backported from Varnish 3).

beresp.backend.name (/vcl/variables/beresp-backend-name/)

The name of the backend this response was fetched from (backported from Varnish 3).

beresp.backend.port (/vcl/variables/beresp-backend-port/)

The port of the backend this response was fetched from (backported from Varnish 3).

beresp.grace (/vcl/variables/beresp-grace/)

Defines how long an object can remain overdue and still have Varnish consider it for grace mode. Fastly has implemented `stale-if-error` as a parallel implementation of `beresp.grace`.

beresp.hipaa (/vcl/variables/beresp-hipaa/)

Specifies that content not be cached in non-volatile memory to help customers meet HIPAA security requirements. See our guide on HIPAA and caching PHI (/guides/compliance/hipaa-and-caching-phi) for instructions on enabling this feature for your account.

beresp.pci (/vcl/variables/beresp-pci/)

Specifies that content be cached in a manner that satisfies PCI DSS requirements. See our PCI compliance description (/guides/detailed-product-descriptions/pci-compliant-caching) for instructions on enabling this feature for your account.

client.port (/vcl/variables/client-port/)

Returns the remote client port. This could be useful as a seed that returns the same value both in an ESI and a top level request. For example, you could hash `client.ip` and `client.port` to get a value used both in ESI and the top level request.

req.grace (/vcl/variables/req-grace/)

Defines how long an object can remain overdue and still have Varnish consider it for grace mode.

req.http.host (/vcl/variables/req-http-host/)

The full host name, without the path or query parameters. For example, in the request `www.example.com/index.html?a=1&b=2`, `req.http.host` will return `www.example.com`.

req.is_ipv6 (/vcl/variables/req-is-ipv6/)

Indicates whether the request was made using IPv6 or not. This is a boolean, read-only variable available in `vcl_recv`, `vcl_hash`, `vcl_deliver` and `vcl_log`.

req.restarts (/vcl/variables/req-restarts/)

Counts the number of times the VCL has been restarted.

req.topurl (/vcl/variables/req-topurl/)

In an ESI subrequest, returns the URL of the top-level request.

req.url.basename (/vcl/variables/req-url-basename/)

The file name specified in a URL. For example, in the request `www.example.com/1/hello.gif?foo=bar`, `req.url.basename` will return `hello.gif`.

req.url.dirname (/vcl/variables/req-url-dirname/)

The directories specified in a URL. For example, in the request `www.example.com/1/hello.gif?foo=bar`, `req.url.dirname` will return `/1`. In the request `www.example.com/5/inner/hello.gif?foo=bar`, `req.url.dirname` will return `/5/inner`.

req.url.ext (/vcl/variables/req-url-ext/)

The file extension specified in a URL. For example, in the request `www.example.com/1/hello.gif?foo=bar`, `req.url.ext` will return `gif`.

req.url.path (/vcl/variables/req-url-path/)

The full path, without any query parameters. For example, in the request `www.example.com/index.html?a=1&b=2`, `req.url.path` will return `/index.html`.

req.url.qs (/vcl/variables/req-url-qs/)

The query string portion of `req.url`. This will be from immediately after the `?` to the end of the URL.

req.url (/vcl/variables/req-url/)

The full path, including query parameters. For example, in the request `www.example.com/index.html?a=1&b=2`, req.url will return `/index.html?a=1&b=2`.

stale.exists (/vcl/variables/stale-exists/)

Specifies if a given object has stale content (/guides/performance-tuning/serving-stale-content) in cache. Returns `TRUE` or `FALSE`.

Query string manipulation (/vcl/query-string-manipulation/)

Examples

In your VCL, you could use `querystring.regfilter_except` as follows:

```
import querystring;

sub vcl_recv {
    # return this URL with only the parameters that match this regular expression
    set req.url = querystring.regfilter_except(req.url, "^(q|p)$");
}
```

You can use `querystring.regfilter` to specify a list of arguments that must not be removed (everything else will be) with a negative look-ahead expression:

```
set req.url = querystring.regfilter(req.url, "^(?!param1|param2)");
```

Query string manipulation Functions

boltsort.sort (/vcl/functions/boltsort-sort/)

Sorts URL parameters. For example, `boltsort.sort("/foo?b=1&a=2&c=3");` returns `"/foo?a=2&b=1&c=3"`.

Format

```
STRING (/vcl/types/string)
boltsort.sort(STRING url)
```

Examples

```
set req.url = boltsort.sort(req.url);
```

querystring.add (/vcl/functions/querystring-add/)

Returns the given URL with the given parameter name and value appended to the end of the query string. The parameter name and value will be URL-encoded when added to the query string.

Format

```
querystring.add(string, string, string)
```

Examples

```
set req.url = querystring.add(req.url, "foo", "bar");
```

querystring.clean (/vcl/functions/querystring-clean/)

Returns the given URL without empty parameters. The query-string is removed if empty (either before or after the removal of empty parameters). Note that a parameter with an empty value does not constitute an empty parameter, so a query string "?something" would not be cleaned.

Format

```
querystring.clean(string)
```

Examples

```
set req.url = querystring.clean(req.url);
```

querystring.filter_except (/vcl/functions/querystring-filter-except/)

Returns the given URL but only keeps the listed parameters.

Format

```
querystring.filter_except(string, string_list)
```

Examples

```
set req.url = querystring.filter_except(req.url,
  "q" + querystring.filtersep() + "p");
```

querystring.filter (/vcl/functions/querystring-filter/)

Returns the given URL without the listed parameters.

Format

```
querystring.filter(string, string_list)
```

Examples

```
set req.url = querystring.filter(req.url,
  "utm_source" + querystring.filtersep() +
  "utm_medium" + querystring.filtersep() +
  "utm_campaign");
```

querystring.filtersep (/vcl/functions/querystring-filtersep/)

Returns the separator needed by the `filter` and `filter_except` functions.

Format

```
querystring.filtersep()
```

querystring.globfilter_except (/vcl/functions/querystring-globfilter-except/)

Returns the given URL but only keeps the parameters matching a glob.

Format

```
querystring.globfilter_except(string, string)
```

Examples

```
set req.url = querystring.globfilter_except(req.url, "sess*");
```

querystring.globfilter (/vcl/functions/querystring-globfilter/)

Returns the given URL without the parameters matching a glob.

Format

```
querystring.globfilter(string, string)
```

Examples

```
set req.url = querystring.globfilter(req.url, "utm_*");
```

querystring.regfilter_except (/vcl/functions/querystring-regfilter-except/)

Returns the given URL but only keeps the parameters matching a regular expression.

Format

```
querystring.regfilter_except(string, string)
```

Examples

```
set req.url = querystring.regfilter_except(req.url, "^(q|p)$");
```

querystring.regfilter (/vcl/functions/querystring-regfilter/)

Returns the given URL without the parameters matching a regular expression.

Format

```
querystring.regfilter(string, string)
```

Examples

```
set req.url = querystring.regfilter(req.url, "^utm_.*");
```

querystring.remove (/vcl/functions/querystring-remove/)

Returns the given URL with its query-string removed.

Format

```
querystring.remove(string)
```

Examples

```
set req.url = querystring.remove(req.url);
```

querystring.set (/vcl/functions/querystring-set/)

Returns the given URL with the given parameter name set to the given value, replacing the original value and removing any duplicates. If the parameter is not present in the query string, the parameter will be appended with the given value to the end of the query string. The parameter name and value will be URL-encoded when set in the query string.

Format

```
querystring.set(string, string, string)
```

Examples

```
set req.url = querystring.set(req.url, "foo", "baz");
```

querystring.sort (/vcl/functions/querystring-sort/)

Returns the given URL with its query-string sorted.

Format

```
querystring.sort(string)
```

Examples

```
set req.url = querystring.sort(req.url);
```

Randomness (/vcl/randomness/)

⚠ **WARNING:** We use BSD random number functions from the GNU C Library (http://www.gnu.org/software/libc/manual/html_node/BSD-Random.html), not true randomizing sources. These VCL functions should not be used for cryptographic (/vcl/cryptographic/) or security purposes.

Random strings

Use the function `randomstr(length [, characters])`. When characters aren't provided, the default will be the 64 characters of `A-Za-z0-9_-`.

```
sub vcl_deliver {
  set resp.http.Foo = "randomstuff=" randomstr(10);
  set resp.http.Bar = "morsecode=" randomstr(50, ".-"); # 50 dots and dashes
}
```

Random content cookies in pure VCL

```
sub vcl_deliver {
  add resp.http.Set-Cookie = "somerandomstuff=" randomstr(10) "; expires=" now + 180d "; path
=/;";
}
```

This adds a cookie named "somerandomstuff" with 10 random characters as value, expiring 180 days from now.

Random decisions

Use the function `randombool( _numerator_, _denominator_ )`, which has a numerator/denominator chance of returning true.

```
sub vcl_recv {
  if (randombool(1, 4)) {
    set req.http.X-AB = "A";
  } else {
    set req.http.X-AB = "B";
  }
}
```

This will add a X-AB header to the request, with a 25% (1 out of 4) chance of having the value "A", and 75% chance of having the value "B".

The `randombool()` function accepts INT function return values, so you could do something this:

```
if (randombool(std.atoi(req.http.Some-Header), 100))  {
  # do something
}
```

Another function, `randombool_seeded()`, takes an additional seed argument. Results for a given seed will always be the same. For instance, in this example the value of the response header will always be `no`:

```
if (randombool_seeded(50, 100, 12345)) {
  set resp.http.Seeded-Value = "yes";
} else {
  set resp.http.Seeded-Value = "no";
}
```

This could be useful for stickiness. For example, if you based the seed off of something that identified a user, you could perform A/B testing without setting a special cookie.

⚠ **WARNING:** The `randombool` and `randombool_seeded` functions do not use secure random numbers and should not be used for cryptographic purposes.

Randomness Functions

randomint_seeded (/vcl/functions/randomint-seeded/)

Identical to randomint (/vcl/functions/randomint/), except takes an additional parameter used to seed the random number generator.

This does not use secure random numbers and should not be used for cryptographic purposes.

Format

```
BOOL (/vcl/types/bool)
randomint_seeded(INTEGER from, INTEGER to, INTEGER seed)
```

Examples

```
if (randomint_seeded(1, 5, user_id) < 5) {
  set req.http.X-ABTest = "A";
} else {
  set req.http.X-ABTest = "B";
}
if (randomint_seeded(-1, 0, 555) == -1) {
  set req.http.X-ABTest = "A";
} else {
  set req.http.X-ABTest = "B";
}
```

randomint (/vcl/functions/randomint/)

Returns a random integer value between `from` and `to`, inclusive.

This does not use secure random numbers and should not be used for cryptographic purposes.

Format

```
BOOL (/vcl/types/bool)
randomint(INTEGER from, INTEGER to)
```

Examples

```
if (randomint(0, 99) < 5) {
  set req.http.X-ABTest = "A";
} else {
  set req.http.X-ABTest = "B";
}
if (randomint(-1, 0) == -1) {
  set req.http.X-ABTest = "A";
} else {
  set req.http.X-ABTest = "B";
}
```

Size (/vcl/size/)

Size Variables

req.body_bytes_read (/vcl/variables/req-body-bytes-read/)

How big the body of a request was in total bytes.

Readable From

- `vcl_deliver`
- `vcl_log`

req.bytes_read (/vcl/variables/req-bytes-read/)

How big a request was in total bytes.

Readable From

- `vcl_deliver`
- `vcl_log`

req.header_bytes_read (/vcl/variables/req-header-bytes-read/)

How big the header of a request was in total bytes.

Readable From

All subroutines

resp.body_bytes_written (/vcl/variables/resp-body-bytes-written/)

How many bytes were written for body of a response.

Readable From

- `vcl_log`

resp.bytes_written (/vcl/variables/resp-bytes-written/)

How many bytes in total were sent as a response.

Readable From

- `vcl_log`

resp.completed (/vcl/variables/resp-completed/)

Whether the response completed successfully or not.

Readable From

- `vcl_log`

resp.header_bytes_written (/vcl/variables/resp-header-bytes-written/)

How many bytes were written for the header of a response.

Readable From

- `vcl_log`

TLS and HTTP/2 (/vcl/tls-and-http2/)

When using these variables, remember the following:

- These variables are currently only allowed to appear within the VCL hooks `vcl_recv`, `vcl_hash`, `vcl_deliver` and `vcl_log`.
- Requests made with HTTP/2 will appear in custom logs (/guides/streaming-logs/custom-log-formats) as HTTP1.1 because those requests will already have been decrypted by the time Varnish sees it. Specifically, the `%r` variable will not accurately represent the type of HTTPX request being processed.

TLS and HTTP/2 Functions

h2.push (/vcl/functions/h2-push/)

Triggers an HTTP/2 server push of the asset passed into the function as the input-string.

Format

```
h2.push(string)
```

TLS and HTTP/2 Variables

fastly_info.h2.is_push (/vcl/variables/fastly-info-h2-is-push/)

Whether or not this request was a server-initiated request generated to create an HTTP/2 Server-pushed response. Returns a boolean value.

Type

`BOOL (/vcl/types/bool)`

Readable From

- `vcl_recv`
- `vcl_hash`
- `vcl_deliver`
- `vcl_log`

fastly_info.h2.stream_id (/vcl/variables/fastly-info-h2-stream-id/)

If the request was made over HTTP/2, the underlying HTTP/2 stream ID.

Readable From

- `vcl_recv`
- `vcl_hash`
- `vcl_deliver`
- `vcl_log`

fastly_info.is_h2 (/vcl/variables/fastly-info-is-h2/)

Whether or not the request was made using http2.

Type

`BOOL (/vcl/types/bool)`

Readable From

- `vcl_recv`
- `vcl_hash`
- `vcl_deliver`
- `vcl_log`

tls.client.cipher (/vcl/variables/tls-client-cipher/)

The cipher suite used to secure the client TLS connection. Example: `"ECDHE-RSA-AES128-GCM-SHA256"`

Readable From

- `vcl_recv`
- `vcl_hash`
- `vcl_deliver`

- `vcl_log`

tls.client.ciphers_sha (/vcl/variables/tls-client-ciphers-sha/)

A SHA1 of the cipher suite identifiers sent from the client as part of the TLS handshake, represented in base64.

Readable From

- `vcl_recv`
- `vcl_hash`
- `vcl_deliver`
- `vcl_log`

tls.client.protocol (/vcl/variables/tls-client-protocol/)

The TLS protocol version this connection is speaking over. Example: `"TLSv1.2"`

Readable From

- `vcl_recv`
- `vcl_hash`
- `vcl_deliver`
- `vcl_log`

tls.client.servername (/vcl/variables/tls-client-servername/)

The Server Name Indication (SNI) the client sent in the `ClientHello` TLS record. Returns `""` if the client did not send SNI. Returns `NULL` (the undefined string) if the request is not a TLS request.

Readable From

- `vcl_recv`
- `vcl_hash`
- `vcl_deliver`
- `vcl_log`

tls.client.tlsexts_sha (/vcl/variables/tls-client-tlsexts-sha/)

A SHA1 of the TLS extension identifiers sent from the client as part of the TLS handshake, represented in base64.

Readable From

- `vcl_recv`
- `vcl_hash`
- `vcl_deliver`
- `vcl_log`

UUID (/vcl/uuid/)

UUID Functions

uuid.dns (/vcl/functions/uuid-dns/)

Returns the RFC4122 (https://tools.ietf.org/html/rfc4122) identifier of DNS namespace, namely the constant `"6ba7b810-9dad-11d1-80b4-00c04fd430c8"`.

Format

```
STRING (/vcl/types/string)
dns()
```

uuid.is_valid (/vcl/functions/uuid-is-valid/)

Returns true if the string holds a textual representation of a valid UUID (per RFC4122 (https://tools.ietf.org/html/rfc4122)). False otherwise.

Format

```
BOOL (/vcl/types/bool)
is_valid(STRING string)
```

Examples

```
if (uuid.is_valid(req.http.X-Unique-Id)) {
  set beresp.http.X-Unique-Id-Valid = "yes";
}
```

uuid.is_version3 (/vcl/functions/uuid-is-version3/)

Returns true if string holds a textual representation of a valid version 3 UUID. False otherwise.

Format

```
BOOL (/vcl/types/bool)
is_version3(STRING string)
```

Examples

```
if (uuid.is_version3(req.http.X-Unique-Id)) {
  set beresp.http.X-Unique-Id-Valid-V3 = "yes";
}
```

uuid.is_version4 (/vcl/functions/uuid-is-version4/)

Returns true if string holds a textual representation of a valid version 4 UUID. False otherwise.

Format

```
BOOL (/vcl/types/bool)
is_version4(STRING string)
```

Examples

```
if (uuid.is_version4(req.http.X-Unique-Id)) {
  set beresp.http.X-Unique-Id-Valid-V4 = "yes";
}
```

uuid.is_version5 (/vcl/functions/uuid-is-version5/)

Returns true if string holds a textual representation of a valid version 5 UUID. False otherwise.

Format

```
BOOL (/vcl/types/bool)
is_version5(STRING string)
```

Examples

```
if (uuid.is_version5(req.http.X-Unique-Id)) {
  set beresp.http.X-Unique-Id-Valid-V5 = "yes";
}
```

uuid.oid (/vcl/functions/uuid-oid/)

Returns the RFC4122 (https://tools.ietf.org/html/rfc4122) identifier of ISO OID namespace, namely the constant `"6ba7b812-9dad-11d1-80b4-00c04fd430c8"`.

Format

```
STRING (/vcl/types/string)
oid()
```

uuid.url (/vcl/functions/uuid-url/)

Returns the RFC4122 (https://tools.ietf.org/html/rfc4122) identifier of URL namespace, namely the constant `"6ba7b811-9dad-11d1-80b4-00c04fd430c8"`.

Format

```
STRING (/vcl/types/string)
url()
```

uuid.version3 (/vcl/functions/uuid-version3/)

Derives a UUID corresponding to `name` within the given `namespace` using MD5 hash function. Namespace itself is identified by a UUID. Name must be in a canonical form appropriate for selected namespace.

> **NOTE:** In principle, names can be arbitrary octet strings. This implementation will, however, truncate at the first NUL byte.

Format

```
STRING (/vcl/types/string)
version3(STRING namespace, STRING name)
```

Examples

```
set req.http.X-Unique-Id = uuid.version3(uuid.dns(), "www.fastly.com");
```

uuid.version4 (/vcl/functions/uuid-version4/)

Returns a UUID based on random number generator output.

Format

```
STRING (/vcl/types/string)
version4()
```

Examples

```
set req.http.X-Unique-Id = uuid.version4();
```

uuid.version5 (/vcl/functions/uuid-version5/)

Derives a UUID corresponding to `name` within the given `namespace` using SHA1 hash function. Namespace itself is identified by a UUID. Name must be in a canonical form appropriate for selected namespace.

NOTE: In principle, names can be arbitrary octet strings. This implementation will, however, truncate at the first NUL byte.

Format

```
STRING (/vcl/types/string)
version5(STRING namespace, STRING name)
```

Examples

```
set req.http.X-Unique-Id = uuid.version5(uuid.dns(), "www.fastly.com");
```

uuid.x500 (/vcl/functions/uuid-x500/)

Returns the RFC4122 (https://tools.ietf.org/html/rfc4122) identifier of X.500 namespace, namely the constant `"6ba7b812-9dad-11d1-80b4-00c04fd430c8"`.

Format

```
STRING (/vcl/types/string)
x500()
```

Guides

§ Custom VCL

Creating custom VCL (/vcl/custom-vcl/creating-custom-vcl/)

Fastly Varnish syntax is specifically compatible with Varnish 2.1.5 (https://varnish-cache.org/docs/2.1). We run a custom version with added functionality and our VCL parser has its own pre-processor. To mix and match Fastly VCL with your custom VCL successfully, remember the following:

- **You can only restart Varnish requests three times.** This limit exists to prevent infinite loops.
- **VCL doesn't take kindly to Windows newlines (line breaks).** It's best to avoid them entirely.
- **It's best to use `curl -X PURGE` to initiate purges via API (/api/purge).** To restrict access to purging, check for the `FASTLYPURGE` method not the `PURGE` method. When you send a request to Varnish to initiate a purge, the HTTP method that you use is "PURGE", but it has already been changed to "FASTLYPURGE" by the time your VCL runs that request.
- **If you override TTLs with custom VCL, your default TTL set in the configuration (/guides/performance-tuning/serving-stale-content) will not be honored** and the expected behavior may change.

Inserting custom VCL in Fastly's VCL boilerplate

⚠ **DANGER:** Include all of the Fastly VCL boilerplate as a template in your custom VCL file, especially the VCL macro lines (they start with `#FASTLY`). VCL macros expand the code into generated VCL. Add your custom code *in between* the different sections as shown in the example unless you specifically intend to override the VCL at that point.

Custom VCL placement example

```
sub vcl_miss {
  # my custom code
  if (req.http.User-Agent ~ "Googlebot") {
    set req.backend = F_special_google_backend;
  }
#FASTLY miss
  return(fetch);
}
```

Fastly's VCL boilerplate

★ **TIP:** If you use the Fastly Image Optimizer, use the image optimization VCL boilerplate (/guides/imageopto-setup-use/image-optimization-vcl-boilerplate) instead.

```
sub vcl_recv {
#FASTLY recv

  if (req.request != "HEAD" && req.request != "GET" && req.request != "FASTLYPURGE") {
    return(pass);
  }

  return(lookup);
}

sub vcl_fetch {
#FASTLY fetch

  if ((beresp.status == 500 || beresp.status == 503) && req.restarts < 1 && (req.request == "GE
T" || req.request == "HEAD")) {
    restart;
  }

  if (req.restarts > 0) {
    set beresp.http.Fastly-Restarts = req.restarts;
  }

  if (beresp.http.Set-Cookie) {
    set req.http.Fastly-Cachetype = "SETCOOKIE";
    return(pass);
  }

  if (beresp.http.Cache-Control ~ "private") {
    set req.http.Fastly-Cachetype = "PRIVATE";
    return(pass);
  }

  if (beresp.status == 500 || beresp.status == 503) {
    set req.http.Fastly-Cachetype = "ERROR";
    set beresp.ttl = 1s;
    set beresp.grace = 5s;
    return(deliver);
  }

  if (beresp.http.Expires || beresp.http.Surrogate-Control ~ "max-age" || beresp.http.Cache-Con
trol ~ "(s-maxage|max-age)") {
    # keep the ttl here
  } else {
    # apply the default ttl
    set beresp.ttl = 3600s;
  }

  return(deliver);
}

sub vcl_hit {
```

```
#FASTLY hit

  if (!obj.cacheable) {
    return(pass);
  }
  return(deliver);
}

sub vcl_miss {
#FASTLY miss
  return(fetch);
}

sub vcl_deliver {
#FASTLY deliver
  return(deliver);
}

sub vcl_error {
#FASTLY error
}

sub vcl_pass {
#FASTLY pass
}

sub vcl_log {
#FASTLY log
}
```

Uploading custom VCL (/vcl/custom-vcl/uploading-custom-vcl/)

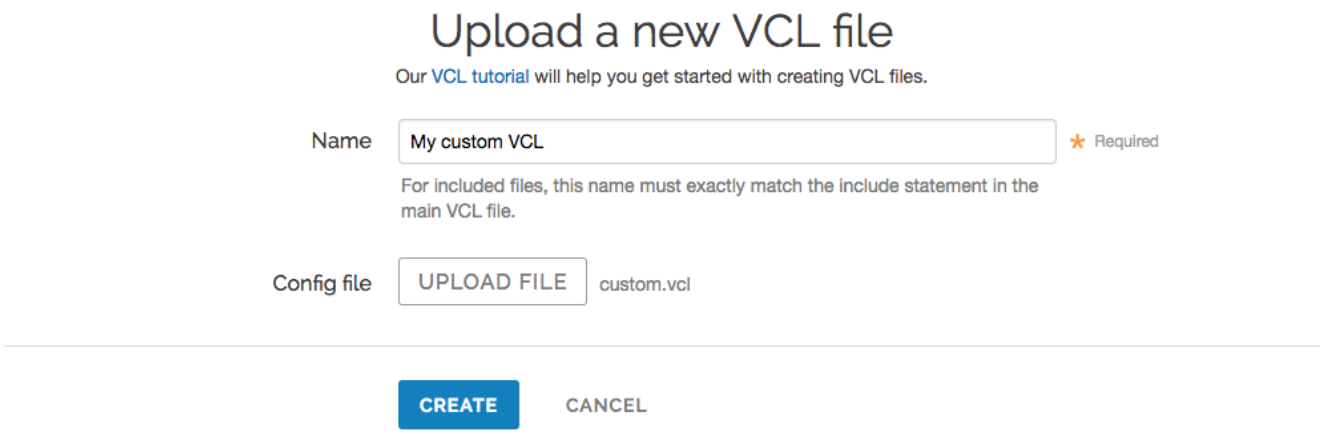
Fastly allows you create your own Varnish Configuration Language (VCL) files with specialized configurations. By uploading custom VCL files, you can use custom VCL and Fastly VCL together at the same time (/vcl/custom-vcl/creating-custom-vcl/). Keep in mind that your custom VCL always takes precedence over VCL generated by Fastly.

IMPORTANT: Personal data should not be incorporated into VCL. Our Compliance and Law FAQ (/guides/compliance-and-law-faq/) describes in detail how Fastly handles personal data privacy.

Uploading a VCL file

Follow these instructions to upload a custom VCL file:

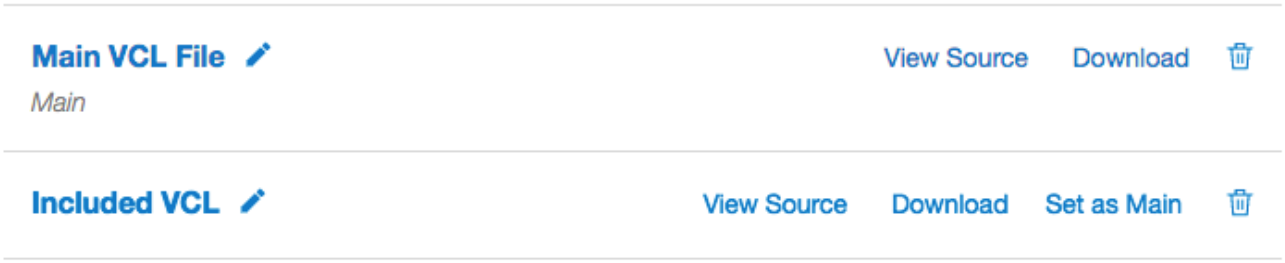
1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **Configuration** button and then select **Clone active**. The Domains page appears.
4. Click the **Custom VCL** tab. The Custom VCL page appears.
5. Click the **Upload a new VCL file** button. The Upload a new VCL file page appears.



6. In the **Name** field, enter the name of the VCL file. For included files, this name must match the include statement in the main VCL file. See how to include additional VCL configurations for more information.
7. Click **Upload file** and select a file to upload. The name of the uploaded file appears next to the button.

> **IMPORTANT:** Don't upload generated VCL that you've downloaded from the Fastly web interface. Instead, edit and then upload a copy of Fastly's VCL boilerplate (/vcl/custom-vcl/creating-custom-vcl#fastlys-vcl-boilerplate) to avoid errors.

8. Click the **Create** button. The VCL file appears in the Varnish Configurations area.

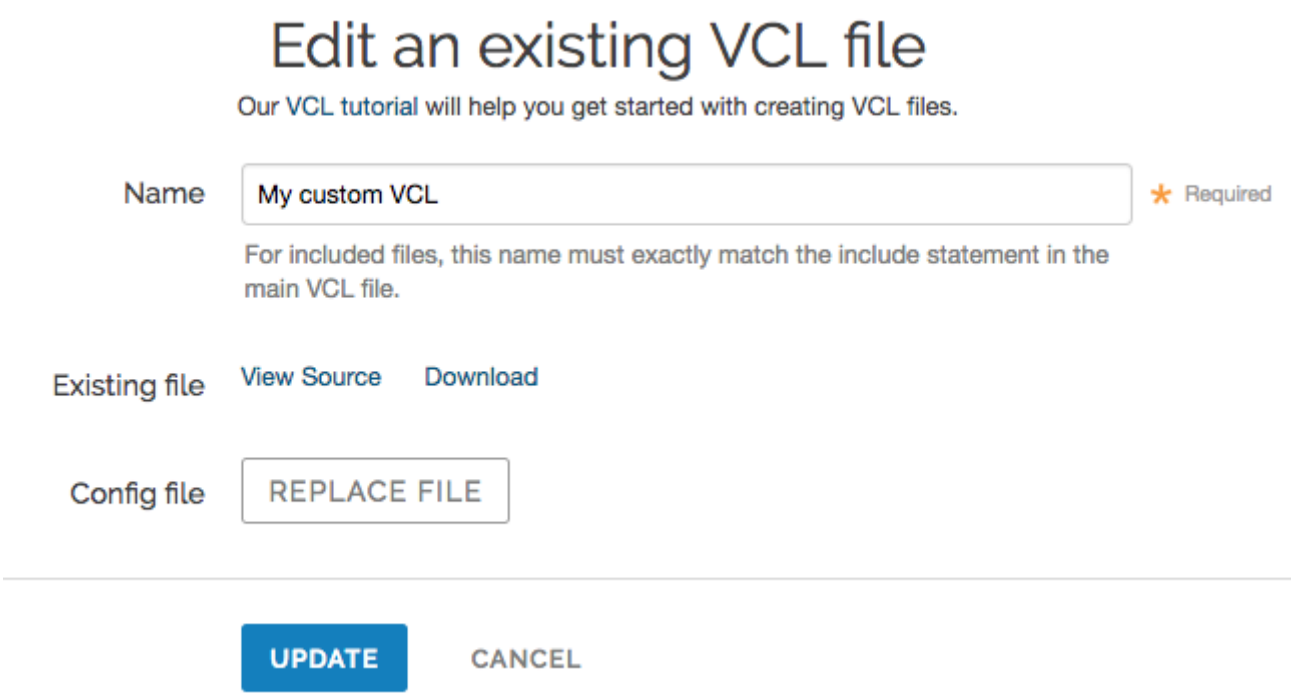


9. Click the **Activate** button to deploy your configuration changes.

Editing a VCL file

To edit an existing VCL file, follow these instructions:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **Configuration** button and then select **Clone active**. The Domains page appears.
4. Click the **Custom VCL** tab. The Custom VCL page appears.
5. In the **Varnish Configurations** area, click the VCL file you want to edit. The Edit an existing VCL file page appears.



6. In the **Name** field, optionally enter a new name of the VCL file.
7. Click the **Download** link to download the appropriate file.
8. Make the necessary changes to your file and save them.
9. Click the **Replace file** button and select the file you updated. The selected file replaces the current VCL file and the file name appears next to the button.
10. Click the **Update** button to update the VCL file in the Fastly application.
11. Click the **Activate** button to deploy your configuration changes.

Including additional VCL configurations

You can apply additional VCL files along with your main VCL by including their file names in the main VCL file using the syntax `include "VCL Name"` where `VCL Name` is the name of an included VCL object you've created.

For example, if you've created an included VCL object called "ACL" (to use an access control list (/guides/access-control-lists/manually-creating-access-control-lists) for code manageability) and the file is named `acl.vcl`, your main VCL configuration file would need to contain this line:

```
include "ACL"
```

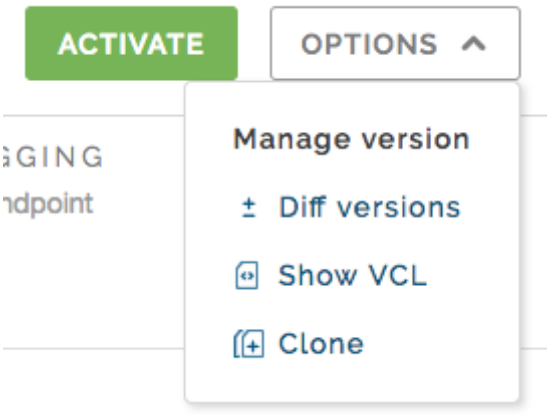
Previewing and testing VCL (/vcl/custom-vcl/previewing-and-testing-vcl/)

Any time you upload VCL files (/vcl/custom-vcl/uploading-custom-vcl/) you can preview and test the VCL prior to activating a new version of your service.

Previewing VCL before activation

To preview VCL prior to activating a service version.

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **Configuration** button and then select **Clone active**. The Domains page appears.
4. Click the **Options** button to open the **Manage version** menu and select **Show VCL**.



The VCL preview page appears.

Testing VCL configurations

You don't need a second account to test your VCL configurations. We recommend adding a new service within your existing account that's specifically designed for testing. A name like "QA" or "testing" or "staging" makes distinguishing between services much easier.

Once created, simply point your testing service to your testing or QA environment. Edit your Fastly configurations for the testing service as if you were creating them for production. Preview your VCL, test things out, and tweak them to get them perfect.

When your testing is complete, make the same changes in your production service that you made to your testing service. If you are using custom VCL, upload the VCL file (/vcl/custom-vcl/uploading-custom-vcl/) to the production service you'll be using.

§ VCL Snippets

About VCL Snippets (/vcl/vcl-snippets/about-vcl-snippets/)

VCL Snippets are short blocks of VCL logic (/guides/vcl-tutorials/guide-to-vcl) that can be included directly in your service configurations. They're ideal for adding small sections of code when you don't need more complex, specialized configurations that sometimes require custom VCL (/vcl/custom-vcl/uploading-custom-vcl/). Fastly supports two types of VCL Snippets:

- **Regular VCL Snippets (/vcl/vcl-snippets/using-regular-vcl-snippets/)** get created as you create versions of your Fastly configurations. They belong to a specific service and any modifications you make to the snippet are locked and deployed when you deploy a new version of that service. You can treat regular snippets like any other Fastly objects because we continue to clone them and deploy them with a service until you specifically delete them. You can create regular snippets using either the web interface or via the API.
- **Dynamic VCL Snippets (/guides/vcl-snippets/using-dynamic-vcl-snippets)** can be modified and deployed any time they're changed. Because they are versionless objects (much like Edge Dictionaries (/guides/edge-dictionaries/) or ACLs (/guides/access-control-lists/) at the edge), dynamic snippets can be modified independently from service changes. This means you can modify snippet code rapidly without deploying a service version that may not be ready for production. You can only create dynamic snippets via the API.

Limitations of VCL Snippets

- Snippets are limited to 1MB in size by default. If you need to store snippets larger than the limit, contact support@fastly.com (mailto:support@fastly.com).
- Snippets do not currently support conditions, though `if` statements can be used within snippet code instead.
- Snippets cannot currently be shared between services.

Using dynamic VCL Snippets (/vcl/vcl-snippets/using-dynamic-vcl-snippets/)

Dynamic VCL Snippets are one of two types of snippets (/vcl/vcl-snippets/about-vcl-snippets) that allow you to insert small sections of VCL logic into your service configuration without requiring custom VCL (/vcl/custom-vcl/uploading-custom-vcl/) (though you can still include snippets in custom VCL when necessary).

You can only create dynamic snippets via the API. Because they are versionless objects (much like Edge Dictionaries (/guides/edge-dictionaries/) or ACLs (/guides/access-control-lists/) at the edge), dynamic snippets can be modified independently from changes to your Fastly service. This means you can modify snippet code rapidly without deploying a service version that may not be ready for production.

Creating and using a dynamic VCL Snippet

Using the cURL command line tool, make the following API call in a terminal application:

```
curl -X POST -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippe
t -H "Fastly-Key:FASTLY_API_TOKEN" -H 'Content-Type: application/x-www-form-urlencoded' --data
$'name=my_dynamic_snippet_name&type=recv&dynamic=1&content=if ( req.url ) {\n set req.http.my-s
nippet-test-header = "true";\n}';
```

Fastly returns a JSON response that looks like this:

```
{
 "service_id": "<Service Id>",
 "version": "<Editable Version>",
 "name": "my_dynamic_snippet_name",
 "type": "recv",
 "priority": 100,
 "dynamic": 1,
 "content": null,
 "id": "decafbad12345",
 "created_at": "2016-09-09T20:34:51+00:00",
 "updated_at": "2016-09-09T20:34:51+00:00",
 "deleted_at": null
}
```

NOTE: The returned JSON includes `"content": null`. This happens because the content is stored in a separate, unversioned object.

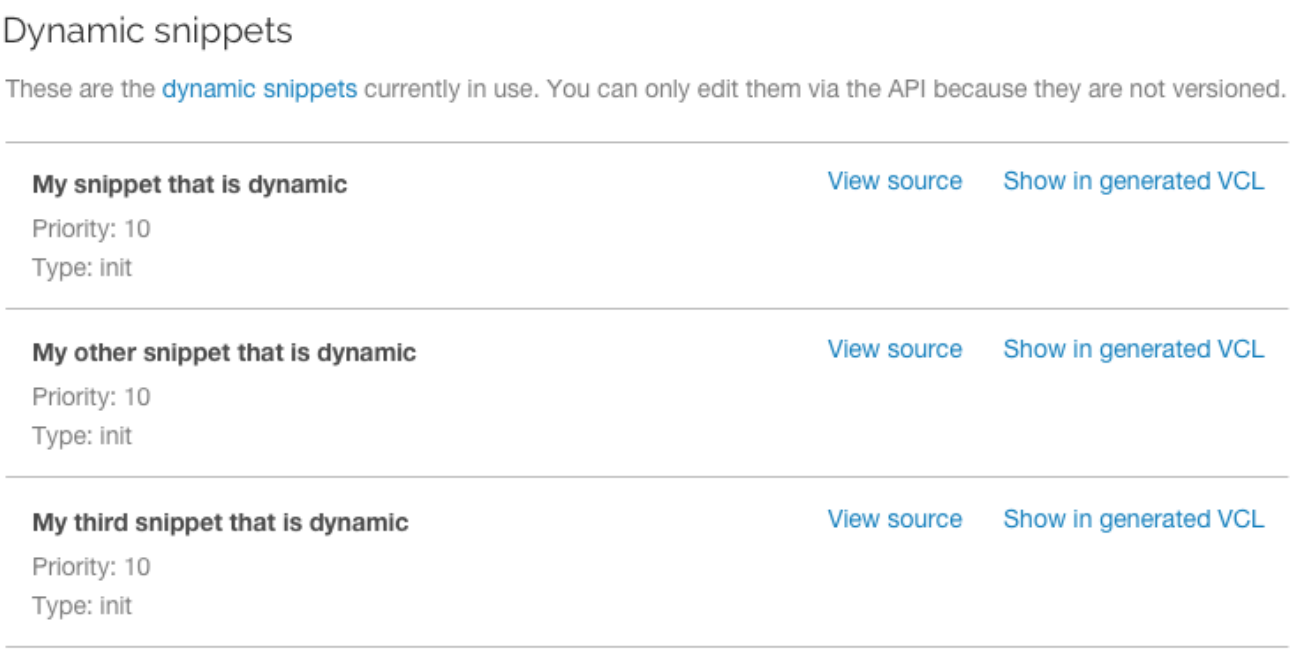
Viewing dynamic VCL Snippets in the web interface

You can view a list of dynamic VCL snippets. You can also view just the source of a specific snippet or a specific snippet's location in generated VCL.

Viewing a list of dynamic VCL Snippets

To view the entire list of a service's dynamic VCL Snippets directly in the web interface:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears listing all dynamic VCL Snippets for your service in the Dynamic snippets area.



Viewing the source of a specific snippet

You can view just the source of a specific snippet:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears.
4. Click the **View Source** link to the right of the name of the snippet. A view source window appears.

Viewing the location of a specific snippet in generated VCL

You can view a specific snippet's location in generated VCL:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears.
4. Click the **Show in Generated VCL** link to the right of the name of the snippet. The Generated VCL window appears.

Fetching a list of all dynamic VCL Snippets

To list all dynamic VCL Snippets attached to a service, make the following API call in a terminal application:

```
curl -X GET -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippet
-H "Fastly-Key:FASTLY_API_TOKEN"
```

Fetching an individual dynamic VCL Snippet

To fetch an individual snippet, make the following API call in a terminal application:

```
curl -X GET -s https://api.fastly.com/service/<Service ID>/snippet/<my_dynamic_snippet_id> -H
"Fastly-Key:FASTLY_API_TOKEN"
```

Unlike fetching regular VCL Snippets (/vcl/vcl-snippets/using-regular-vcl-snippets#fetching-an-individual-regular-vcl-snippet), you do not include the version in the URL and you must use the ID returned when the snippet was created, not the name.

Updating an existing dynamic VCL Snippet

To update an individual snippet, make the following API call in a terminal application:

```
curl -X PUT -s https://api.fastly.com/service/<Service ID>/snippet/<my_dynamic_snippet_id> -H
"Fastly-Key:FASTLY_API_TOKEN" -H 'Content-Type: application/x-www-form-urlencoded' --data $'con
tent=if ( req.url ) {\n set req.http.my-snippet-test-header = \"affirmative\";\n}';
```

Deleting an existing dynamic VCL Snippet

To delete an individual snippet, make the following API call in a terminal application:

```
curl -X DELETE -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snip
pet/<my_dynamic_snippet_name> -H "Fastly-Key:FASTLY_API_TOKEN"
```

Including dynamic snippets in custom VCL

By specifying a location of `none` for the `type` parameter, snippets will not be rendered in VCL. This allows you to include snippets in custom VCL using the following syntax:

```
include "snippet::<snippet name>"
```

The same VCL Snippet can be included in custom VCL in as many places as needed.

Example use: blocking site scrapers

Say you wanted to implement some pattern matching against incoming requests to block someone trying to scrape your site. Say also that you've developed a system that looks at all incoming requests and generates a set of rules that can identify scrapers using a combination of the incoming IP address, the browser, and the URL they're trying to fetch. Finally, say that the system updates the rules every 20 minutes.

If, during system updates, your colleagues are also making changes to the rest of your Fastly configuration, you probably don't want the system to automatically deploy the latest version of the service since it might be untested. Instead you could generate the rules as a Dynamic VCL Snippet. Whenever the snippet is updated, all other logic remains the same as the currently deployed version and only your rules are modified.

Using regular VCL Snippets (/vcl/vcl-snippets/using-regular-vcl-snippets/)

Regular VCL Snippets are one of two types of snippets (/vcl/vcl-snippets/about-vcl-snippets) that allow you to insert small sections of VCL logic into your service configuration without requiring custom VCL (/vcl/custom-vcl/uploading-custom-vcl/) (though you can still include snippets in custom VCL when necessary).

Unlike dynamic snippets (/vcl/vcl-snippets/using-dynamic-vcl-snippets), regular snippets can be created via the web interface or via the API. They are considered "versioned" objects. They belong to a specific service and any modifications you make to the snippet are locked and deployed when you deploy a new version of that service. We continue to clone them and deploy them with a service until you specifically delete them.

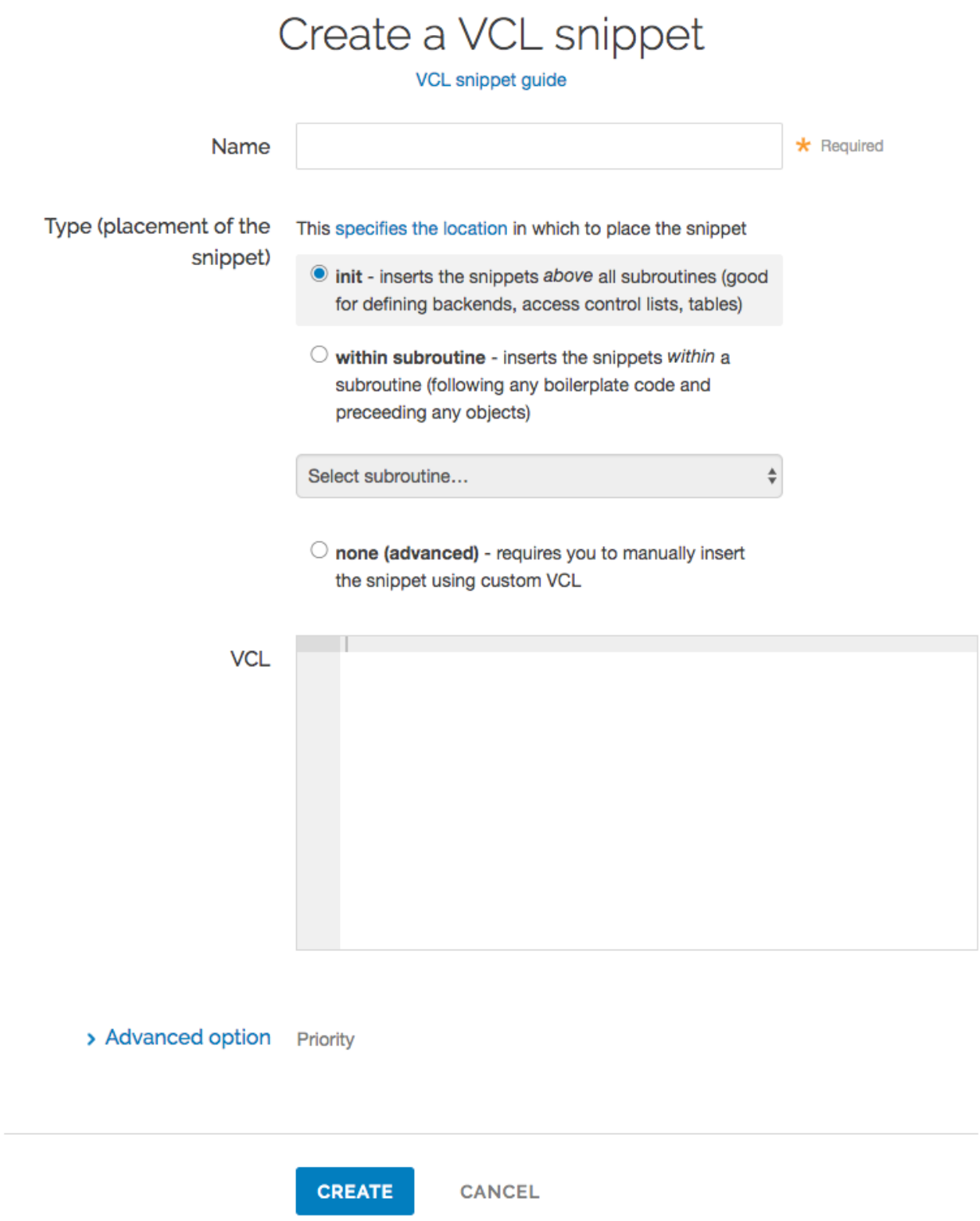
Creating a regular VCL Snippet

You can create regular VCL Snippets via the web interface or via the API.

Via the web interface

To create a regular VCL Snippet via the web interface:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears.
4. Click **Create Snippet**. The Create a VCL snippet page appears.



5. In the **Name** field, type an appropriate name (for example, `Example Snippet`).
6. Using the **Type** controls, select the location in which the snippet should be placed as follows:
   - Select `init` to insert it above all subroutines in your VCL.

- Select `within subroutine` to insert it within a specific subroutine and then select the specific subroutine from the **Select subroutine** menu.
- Select `none (advanced)` to insert it manually. See Including regular snippets in custom VCL (/vcl/vcl-snippets/using-regular-vcl-snippets#including-regular-snippets-in-custom-vcl) for the additional manual insertion requirements if you select this option.

7. In the **VCL** field, type the snippet of VCL logic to be inserted for your service version.
8. Click **Create** to create the snippet.

Via the API

To create a regular VCL Snippet via the API, make the following API call using the cURL command line tool in a terminal application:

```
curl -X POST -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippe
t -H "Fastly-Key:FASTLY_API_TOKEN" -H `fastly-cookie` -H 'Content-Type: application/x-www-form-
urlencoded' --data $'name=my_regular_snippet&type=recv&dynamic=0&content=if ( req.url ) {\n set
req.http.my-snippet-test-header = "true";\n}';
```

Fastly returns a JSON response that looks like this:

```
{
 "service_id": "<Service Id>",
 "version": "<Editable Version>",
 "name": "my_regular_snippet",
 "type": "recv",
 "content": "if ( req.url ) {\n set req.http.my-snippet-test-header = \"true\";\n}",
 "priority": 100,
 "dynamic": 0,
 "id": "56789exampleid",
 "created_at": "2016-09-09T20:34:51+00:00",
 "updated_at": "2016-09-09T20:34:51+00:00",
 "deleted_at": null
}
```

NOTE: When regular VCL snippets get created, an `id` field will be returned that isn't used. The field only applies to dynamic VCL Snippets (/vcl/vcl-snippets/using-dynamic-vcl-snippets). In addition, the returned JSON includes a populated `content` field because the snippet content is stored in a versioned object.

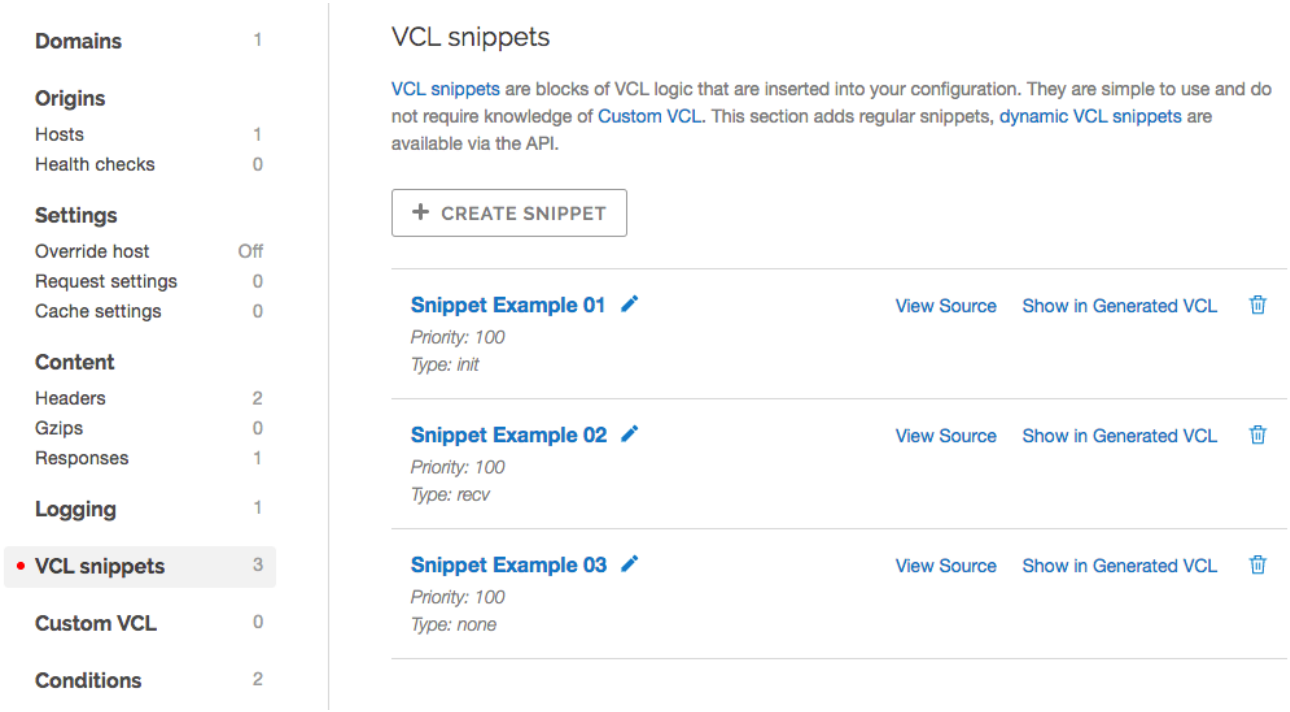
Viewing regular VCL Snippets in the web interface

You can view a list of regular VCL snippets. You can also view just the source of a specific snippet or a specific snippet's location in generated VCL.

Viewing a list of regular VCL Snippets

To view the entire list of a service's regular VCL Snippets directly in the web interface:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears listing all available VCL snippets for your service.



Viewing the source of a specific snippet

You can view just the source of a specific snippet:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears.
4. Click the **View Source** link to the right of the name of the snippet. A view source window appears.

Viewing the location of a specific snippet in generated VCL

You can view a specific snippet's location in generated VCL:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears.

4. Click the **Show in Generated VCL** link to the right of the name of the snippet. The Generated VCL window appears.

Fetching regular VCL Snippets via the API

You can fetch regular VCL Snippets for a particular service via the API either singly or all at once.

Fetching an individual regular VCL Snippet

To fetch an individual snippet, make the following API call in a terminal application:

```
curl -X GET -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippe
t/<Snippet Name e.g my_regular_snippet> -H "Fastly-Key:FASTLY_API_TOKEN"
```

Unlike fetching dynamic VCL Snippets (/vcl/vcl-snippets/using-dynamic-vcl-snippets#fetching-an-individual-dynamic-vcl-snippet) you include the version in the URL and you must use the name of the snippet, not the ID.

Fetching a list of regular VCL Snippets

To list all regular VCL Snippets attached to a service, make the following API call in a terminal application:

```
curl -X GET -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippe
t/ -H "Fastly-Key:FASTLY_API_TOKEN"
```

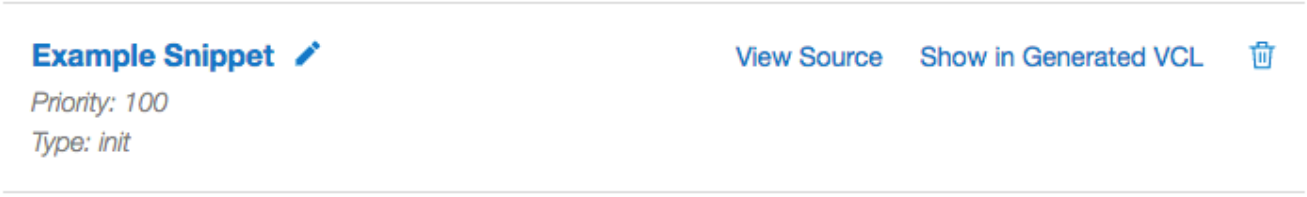
Updating an existing regular VCL Snippet

You can update existing regular VCL Snippets via the web interface or via the API.

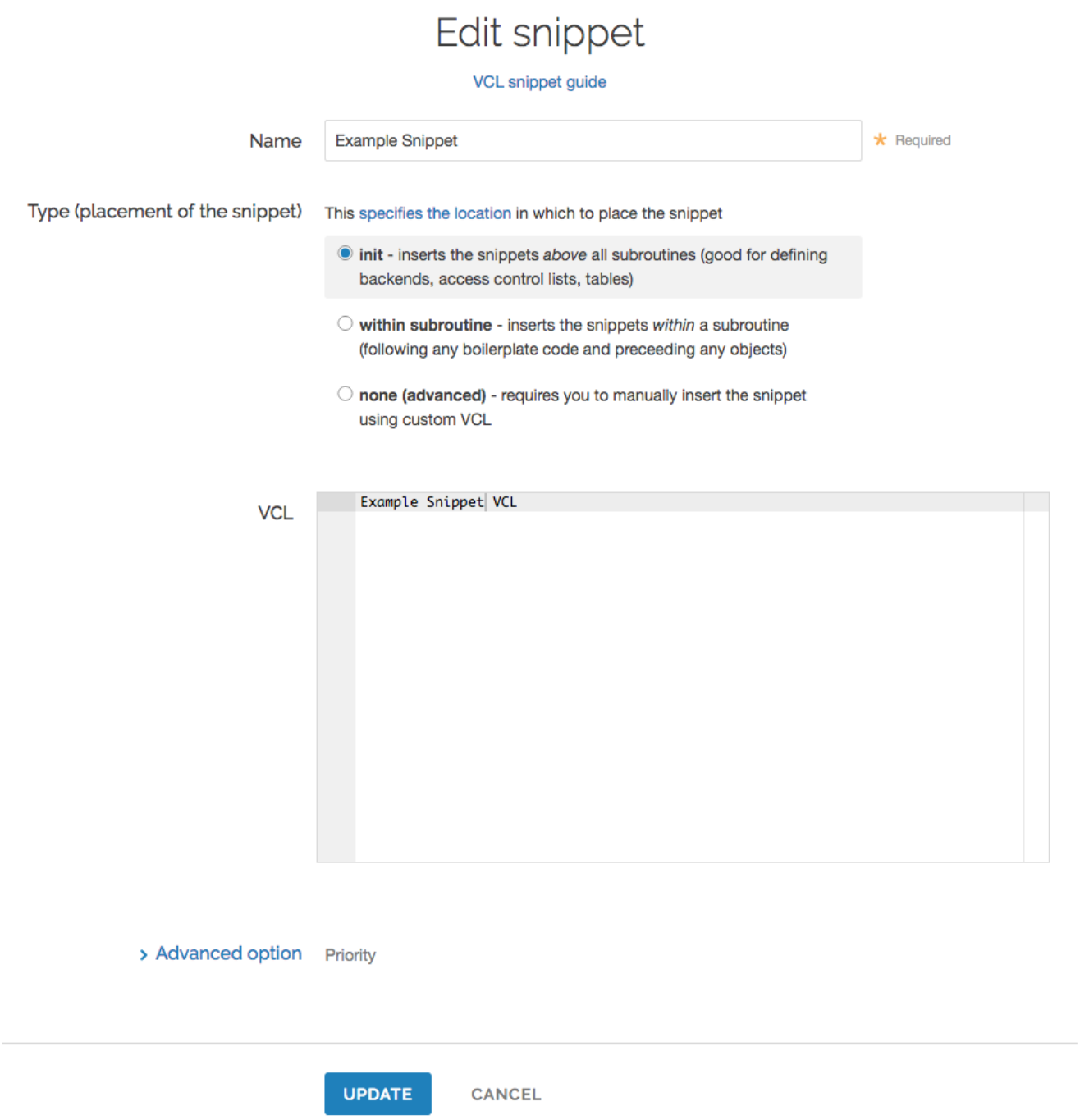
Via the web interface

To update an individual snippet via the web interface:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears.
4. Click the pencil icon next to the name of the snippet to be updated.



The Edit snippet page appears.



5. Update the snippet's settings or VCL as appropriate.
6. Click **Update** to save your changes.

Via the API

To update an individual snippet via the API, make the following API call in a terminal application:

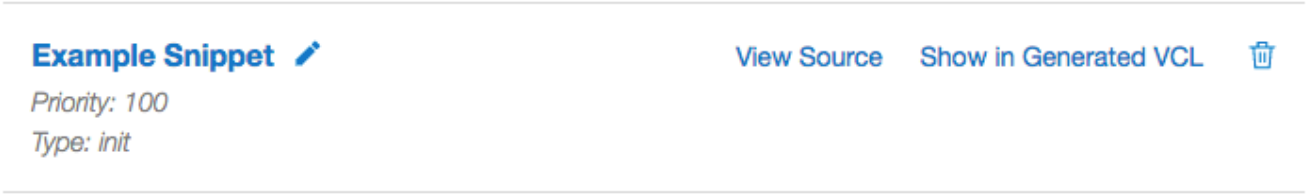
```
curl -X PUT -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippe
t/<Snippet Name e.g my_regular_snippet> -H "Fastly-Key:FASTLY_API_TOKEN" -H 'Content-Type: appl
ication/x-www-form-urlencoded' --data $'content=if ( req.url ) {\n set req.http.my-snippet-test
-header = \"affirmative\";\n}';
```

Deleting an existing regular VCL Snippet

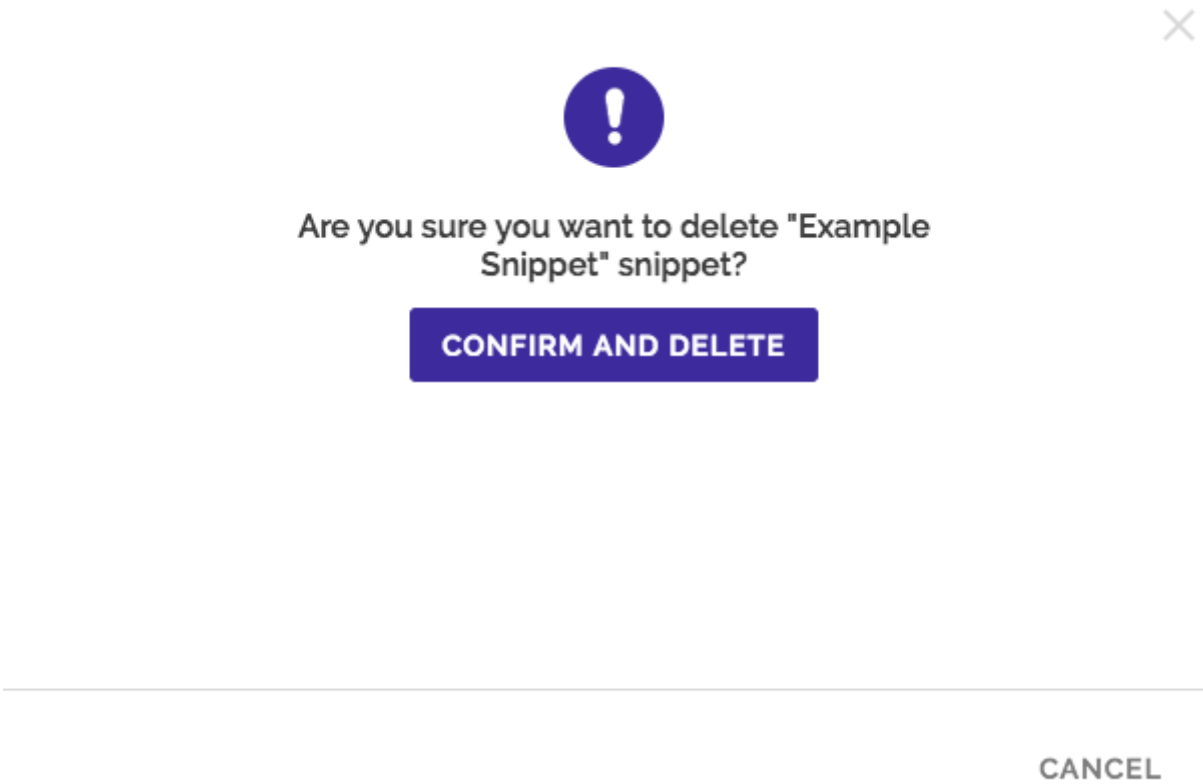
You can update existing regular VCL Snippets via the web interface or via the API.

Via the web interface

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears.
4. Click the trashcan icon to the right of the name of the snippet to be updated.



A confirmation window appears.



5. Click **Confirm and Delete**.

Via the API

To delete an individual snippet via the API, make the following API call in a terminal application:

```
curl -X DELETE -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snip
pet/<Snippet Name e.g my_regular_snippet> -H "Fastly-Key:FASTLY_API_TOKEN"
```

Including regular snippets in custom VCL

Snippets will not be rendered in VCL if you select `none (advanced)` for the snippet type in the web interface or specify a location of `none` for the `type` parameter in the API. This allows you to manually include snippets in custom VCL using the following syntax:

`include "snippet::<snippet name>"`

The same VCL Snippet can be included in custom VCL in as many places as needed.

Example use: location-based redirection

Say that you work at a large content publisher and you want to redirect users to different editions of your publication depending on which country their request comes from. Say also that you want the ability to override the edition you deliver to them based on a cookie.

Using regular VCL snippets, you could add a new object with the relevant VCL as follows:

```
if (req.http.Cookie:edition == "US" || client.geo.country_code == "US" || ) {
  set req.http.Edition = "US";
  set req.backend = F_US;
} elseif (req.http.Cookie:edition == "Europe" || server.region ~ "^EU-" ) {
  set req.http.Edition = "EU";
  set req.backend = F_European;
} else {
  set req.http.Edition = "INT";
  set req.backend = F_International;
}
```

This would create an Edition header in VCL, but allow you to override it by setting a condition. You would add the Edition header into Vary (https://www.fastly.com/blog/best-practices-using-vary-header) and then add a false condition (/guides/conditions/using-conditions#using-operators-to-perform-matches-on-complex-logical-expressions) (e.g., `!reg.url`) to your other backends to ensure the correct edition of your publication gets delivered (Remember: VCL Snippets get added to VCL before backends are set.)

§ VCL Reference

Functions (/vcl/functions/)

These VCL functions are supported by Fastly.

Cryptographic (/vcl/cryptographic/)

Fastly provides several functions in VCL (/guides/vcl-tutorials/) for cryptographic- and hashing-related purposes. It is based very heavily on Kristian Lyngstøl's digest vmod (https://github.com/varnish/libvmod-digest) for Varnish 3 (which means you can also refer to that documentation for more detail).

- digest.awsv4_hmac (/vcl/functions/digest-awsv4-hmac/)
- digest.base64_decode (/vcl/functions/digest-base64-decode/)
- digest.base64 (/vcl/functions/digest-base64/)
- digest.base64url_decode (/vcl/functions/digest-base64url-decode/)
- digest.base64url_nopad_decode (/vcl/functions/digest-base64url-nopad-decode/)
- digest.base64url_nopad (/vcl/functions/digest-base64url-nopad/)
- digest.base64url (/vcl/functions/digest-base64url/)
- digest.hash_crc32 (/vcl/functions/digest-hash-crc32/)
- digest.hash_crc32b (/vcl/functions/digest-hash-crc32b/)
- digest.hash_md5 (/vcl/functions/digest-hash-md5/)
- digest.hash_sha1 (/vcl/functions/digest-hash-sha1/)
- digest.hash_sha224 (/vcl/functions/digest-hash-sha224/)
- digest.hash_sha256 (/vcl/functions/digest-hash-sha256/)
- digest.hash_sha384 (/vcl/functions/digest-hash-sha384/)
- digest.hash_sha512 (/vcl/functions/digest-hash-sha512/)
- digest.hmac_md5_base64 (/vcl/functions/digest-hmac-md5-base64/)
- digest.hmac_md5 (/vcl/functions/digest-hmac-md5/)
- digest.hmac_sha1_base64 (/vcl/functions/digest-hmac-sha1-base64/)
- digest.hmac_sha1 (/vcl/functions/digest-hmac-sha1/)

- digest.hmac_sha256_base64 (/vcl/functions/digest-hmac-sha256-base64/)
- digest.hmac_sha256 (/vcl/functions/digest-hmac-sha256/)
- digest.rsa_verify (/vcl/functions/digest-rsa-verify/)
- digest.secure_is_equal (/vcl/functions/digest-secure-is-equal/)
- digest.time_hmac_md5 (/vcl/functions/digest-time-hmac-md5/)
- digest.time_hmac_sha1 (/vcl/functions/digest-time-hmac-sha1/)
- digest.time_hmac_sha256 (/vcl/functions/digest-time-hmac-sha256/)

Date and time (/vcl/date-and-time/)

By default VCL includes the `now` variable, which provides the current time (for example, `Wed, 17 Sep 2025 23:19:06 GMT`). Fastly adds several new Varnish variables and functions that allow more flexibility when dealing with dates and times.

- std.integer2time (/vcl/functions/std-integer2time/)
- std.time (/vcl/functions/std-time/)
- strftime (/vcl/functions/strftime/)
- time.add (/vcl/functions/time-add/)
- time.hex_to_time (/vcl/functions/time-hex-to-time/)
- time.is_after (/vcl/functions/time-is-after/)
- time.sub (/vcl/functions/time-sub/)

Miscellaneous (/vcl/miscellaneous/)

Fastly has added several miscellaneous features to Varnish that don't easily fit into specific categories.

- cstr_escape (/vcl/functions/cstr-escape/)
- http_status_matches (/vcl/functions/http-status-matches/)
- if() (/vcl/functions/if/)
- std.atoi (/vcl/functions/std-atoi/)
- std.ip (/vcl/functions/std-ip/)
- std.ip2str (/vcl/functions/std-ip2str/)
- std.str2ip (/vcl/functions/std-str2ip/)
- std.strlen (/vcl/functions/std-strlen/)
- std.strstr (/vcl/functions/std-strstr/)

- std.strtol (/vcl/functions/std-strtol/)
- std.tolower (/vcl/functions/std-tolower/)
- std.toupper (/vcl/functions/std-toupper/)
- subfield (/vcl/functions/subfield/)
- urldecode (/vcl/functions/urldecode/)
- urlencode (/vcl/functions/urlencode/)

Query string manipulation (/vcl/query-string-manipulation/)

Fastly provides a number of extensions to VCL (/guides/vcl-tutorials/guide-to-vcl#fastlys-vcl-extensions), including several functions for query-string manipulation based on Dridi Boukelmoune's vmod-querystring (https://github.com/Dridi/libvmod-querystring) for Varnish.

- boltsort.sort (/vcl/functions/boltsort-sort/)
- querystring.add (/vcl/functions/querystring-add/)
- querystring.clean (/vcl/functions/querystring-clean/)
- querystring.filter_except (/vcl/functions/querystring-filter-except/)
- querystring.filter (/vcl/functions/querystring-filter/)
- querystring.filtersep (/vcl/functions/querystring-filtersep/)
- querystring.globfilter_except (/vcl/functions/querystring-globfilter-except/)
- querystring.globfilter (/vcl/functions/querystring-globfilter/)
- querystring.regfilter_except (/vcl/functions/querystring-regfilter-except/)
- querystring.regfilter (/vcl/functions/querystring-regfilter/)
- querystring.remove (/vcl/functions/querystring-remove/)
- querystring.set (/vcl/functions/querystring-set/)
- querystring.sort (/vcl/functions/querystring-sort/)

Randomness (/vcl/randomness/)

Fastly exposes a number of functions that support the insertion of random strings, content cookies, and decisions into requests.

- randomint_seeded (/vcl/functions/randomint-seeded/)
- randomint (/vcl/functions/randomint/)

TLS and HTTP/2 (/vcl/tls-and-http2/)

Fastly has added several variables that expose information about the TLS and HTTP/2 attributes of a request.

- h2.push (/vcl/functions/h2-push/)

UUID (/vcl/uuid/)

The universally unique identifier (UUID) module provides interfaces for generating and validating unique identifiers as defined by RFC4122 (https://tools.ietf.org/html/rfc4122). Version 1 identifiers, based on current time and host identity, are currently not supported.

- uuid.dns (/vcl/functions/uuid-dns/)
- uuid.is_valid (/vcl/functions/uuid-is-valid/)
- uuid.is_version3 (/vcl/functions/uuid-is-version3/)
- uuid.is_version4 (/vcl/functions/uuid-is-version4/)
- uuid.is_version5 (/vcl/functions/uuid-is-version5/)
- uuid.oid (/vcl/functions/uuid-oid/)
- uuid.url (/vcl/functions/uuid-url/)
- uuid.version3 (/vcl/functions/uuid-version3/)
- uuid.version4 (/vcl/functions/uuid-version4/)
- uuid.version5 (/vcl/functions/uuid-version5/)
- uuid.x500 (/vcl/functions/uuid-x500/)

Variables (/vcl/variables/)

These VCL variables are supported by Fastly.

Date and time (/vcl/date-and-time/)

By default VCL includes the `now` variable, which provides the current time (for example, `Wed, 17 Sep 2025 23:19:06 GMT`). Fastly adds several new Varnish variables and functions that allow more flexibility when dealing with dates and times.

- now.sec (/vcl/variables/now-sec/)
- now (/vcl/variables/now/)
- time.elapsed.msec_frac (/vcl/variables/time-elapsed-msec-frac/)

- time.elapsed.msec (/vcl/variables/time-elapsed-msec/)
- time.elapsed.sec (/vcl/variables/time-elapsed-sec/)
- time.elapsed.usec_frac (/vcl/variables/time-elapsed-usec-frac/)
- time.elapsed.usec (/vcl/variables/time-elapsed-usec/)
- time.elapsed (/vcl/variables/time-elapsed/)
- time.end.msec_frac (/vcl/variables/time-end-msec-frac/)
- time.end.msec (/vcl/variables/time-end-msec/)
- time.end.sec (/vcl/variables/time-end-sec/)
- time.end.usec_frac (/vcl/variables/time-end-usec-frac/)
- time.end.usec (/vcl/variables/time-end-usec/)
- time.end (/vcl/variables/time-end/)
- time.start.msec_frac (/vcl/variables/time-start-msec-frac/)
- time.start.msec (/vcl/variables/time-start-msec/)
- time.start.sec (/vcl/variables/time-start-sec/)
- time.start.usec_frac (/vcl/variables/time-start-usec-frac/)
- time.start.usec (/vcl/variables/time-start-usec/)
- time.start (/vcl/variables/time-start/)
- time.to_first_byte (/vcl/variables/time-to-first-byte/)

Geolocation (/vcl/geolocation/)

Fastly exposes a number of geographic variables for you to take advantage of inside VCL for both IPv4 and IPv6 client IPs.

- client.as.name (/vcl/variables/client-as-name/)
- client.as.number (/vcl/variables/client-as-number/)
- client.geo.area_code (/vcl/variables/client-geo-area-code/)
- client.geo.city.ascii (/vcl/variables/client-geo-city-ascii/)
- client.geo.city.utf8 (/vcl/variables/client-geo-city-utf8/)
- client.geo.city (/vcl/variables/client-geo-city/)
- client.geo.conn_speed (/vcl/variables/client-geo-conn-speed/)
- client.geo.continent_code (/vcl/variables/client-geo-continent-code/)
- client.geo.country_code (/vcl/variables/client-geo-country-code/)

- client.geo.country_code3 (/vcl/variables/client-geo-country-code3/)
- client.geo.country_name.ascii (/vcl/variables/client-geo-country-name-ascii/)
- client.geo.country_name.ascii (/vcl/variables/client-geo-country-name-utf8/)
- client.geo.country_name (/vcl/variables/client-geo-country-name/)
- client.geo.gmt_offset (/vcl/variables/client-geo-gmt-offset/)
- client.geo.latitude (/vcl/variables/client-geo-latitude/)
- client.geo.longitude (/vcl/variables/client-geo-longitude/)
- client.geo.metro_code (/vcl/variables/client-geo-metro-code/)
- client.geo.postal_code (/vcl/variables/client-geo-postal-code/)
- client.geo.region (/vcl/variables/client-geo-region/)
- server.datacenter (/vcl/variables/server-datacenter/)
- server.region (/vcl/variables/server-region/)

Miscellaneous (/vcl/miscellaneous/)

Fastly has added several miscellaneous features to Varnish that don't easily fit into specific categories.

- bereq.url.basename (/vcl/variables/bereq-url-basename/)
- bereq.url.dirname (/vcl/variables/bereq-url-dirname/)
- bereq.url.qs (/vcl/variables/bereq-url-qs/)
- bereq.url (/vcl/variables/bereq-url/)
- beresp.backend.ip (/vcl/variables/beresp-backend-ip/)
- beresp.backend.name (/vcl/variables/beresp-backend-name/)
- beresp.backend.port (/vcl/variables/beresp-backend-port/)
- beresp.grace (/vcl/variables/beresp-grace/)
- beresp.hipaa (/vcl/variables/beresp-hipaa/)
- beresp.pci (/vcl/variables/beresp-pci/)
- client.port (/vcl/variables/client-port/)
- req.grace (/vcl/variables/req-grace/)
- req.http.host (/vcl/variables/req-http-host/)
- req.is_ipv6 (/vcl/variables/req-is-ipv6/)
- req.restarts (/vcl/variables/req-restarts/)

- req.topurl (/vcl/variables/req-topurl/)
- req.url.basename (/vcl/variables/req-url-basename/)
- req.url.dirname (/vcl/variables/req-url-dirname/)
- req.url.ext (/vcl/variables/req-url-ext/)
- req.url.path (/vcl/variables/req-url-path/)
- req.url.qs (/vcl/variables/req-url-qs/)
- req.url (/vcl/variables/req-url/)
- stale.exists (/vcl/variables/stale-exists/)

Size (/vcl/size/)

To allow better reporting, Fastly has added several variables to VCL to give more insight into what happened in a request.

- req.body_bytes_read (/vcl/variables/req-body-bytes-read/)
- req.bytes_read (/vcl/variables/req-bytes-read/)
- req.header_bytes_read (/vcl/variables/req-header-bytes-read/)
- resp.body_bytes_written (/vcl/variables/resp-body-bytes-written/)
- resp.bytes_written (/vcl/variables/resp-bytes-written/)
- resp.completed (/vcl/variables/resp-completed/)
- resp.header_bytes_written (/vcl/variables/resp-header-bytes-written/)

TLS and HTTP/2 (/vcl/tls-and-http2/)

Fastly has added several variables that expose information about the TLS and HTTP/2 attributes of a request.

- fastly_info.h2.is_push (/vcl/variables/fastly-info-h2-is-push/)
- fastly_info.h2.stream_id (/vcl/variables/fastly-info-h2-stream-id/)
- fastly_info.is_h2 (/vcl/variables/fastly-info-is-h2/)
- tls.client.cipher (/vcl/variables/tls-client-cipher/)
- tls.client.ciphers_sha (/vcl/variables/tls-client-ciphers-sha/)
- tls.client.protocol (/vcl/variables/tls-client-protocol/)
- tls.client.servername (/vcl/variables/tls-client-servername/)
- tls.client.tlsexts_sha (/vcl/variables/tls-client-tlsexts-sha/)

Local variables (/vcl/local-variables/)

Fastly VCL supports variables for storing temporary values during request processing. Variables can only be used in the same function block where they were declared. Variables with the same name in different function blocks are unrelated variables.

Declaring a variable

Variables must be declared before they are used, usually at the beginning of the function before any statements. Variables start with `var.` and their names consist of characters in the set `[A-Za-z0-9._-]`. (`:` is explicitly disallowed.)

Declaration syntax is: `declare local var.<name> <type>;`

Variable types

Variables can be of the following types:

- `BOOL` (/vcl/types/bool/)
- `INTEGER` (/vcl/types/integer/)
- `FLOAT` (/vcl/types/float/)
- `TIME` (/vcl/types/time/) (absolute time)
- `RTIME` (/vcl/types/rtime/) (relative time)
- `STRING` (/vcl/types/string/)

Declared variables are initialized to the zero value of the type:

- `0` for numeric types
- `false` for `BOOL`
- `NULL` for `STRING`

Usage

Boolean variables

Boolean assignments support boolean variables on the right-hand side as well as `BOOL`-returning functions, conditional expressions, and the `true` and `false` constants.

```
declare local var.boolean BOOL;

# BOOL assignment with RHS variable
set var.boolean = true;
set req.esi = var.boolean;
set resp.http.Bool = if(req.esi, "y", "n");

# BOOL assignment with RHS function
set var.boolean = http_status_matches(resp.status, "200,304");

# BOOL assigment with RHS conditional
set var.boolean = (req.url == "/");

# non-NULL-ness check, like 'if (req.http.Foo) { ... }'
set var.boolean = (req.http.Foo);
```

Numeric variables

Numeric assignment and comparison support numeric variables (anything except `STRING` or `BOOL`) on the right-hand side, including conversion in both directions between `FLOAT` and `INTEGER` types, rounding to the nearest integer in the `FLOAT` to `INTEGER` case.

Invalid conditions or domain errors like division by 0 will set `fastly.error`.

```
declare local var.integer INTEGER;
declare local var.float FLOAT;

# Numeric assignment with RHS variable and
# implicit string conversion for header
set var.integer = req.bytes_read;
set var.integer -= req.body_bytes_read;
set resp.http.VarInteger = var.integer;

# Numeric comparison with RHS variable
set resp.http.VarIntegerOK = if(req.header_bytes_read == var.integer, "y", "n");
```

String variables

String assignments support string concatenation on the right-hand side.

```
declare local var.restarted STRING;

# String concatenation on RHS
set var.restarted = "Request " if(req.restarts > 0, "has", "has not") " restarted.";
```

Time variables

Time variables support both relative and absolute times.

```
declare local var.time TIME;
declare local var.rtime RTIME;

set req.grace = 72s;
set var.rtime = req.grace;
set resp.http.VarRTime = var.rtime;

set var.time = std.time("Fri, 10 Jun 2016 00:02:12 GMT", now);
set var.time -= var.rtime;
# implicit string conversion for header
set resp.http.VarTime = var.time;
```

Operators (/vcl/operators/)

Fastly's VCL provides various arithmetic and conditional operators. Operators are syntactic items which evaluate to a value. Syntax is given in a BNF-like form with the following conventions:

- […] Square brackets enclose an optional item,
- "!" Literal spellings (typically punctuation) are indicated in quotes,
- CNUM Lexical terminals are given in uppercase,
- INTEGER Types are also given in uppercase,
- numeric-expr Grammatical productions are given in lowercase.

Where a binary operator is provided, not all types are implemented on either side. This is a limitation of the current implementation. The following placeholder grammatical clauses are used in this document to indicate which types are valid operands. These are not precisely defined until the grammar has been formally specified, and are intended as a guide for operator context only.

- variable - A variable name
- acl - An ACL name
- expr - An expression of any type
- numeric-expr - An expression evaluating to INTEGER, FLOAT, RTIME, or another numeric type
- time-expr - An expression evaluating to TIME
- assignment-expr - An expression suitable for assignment to a variable by `set`
- conditional-expr - An expression evaluating to BOOL suitable for use with `if` conditions
- string-expr - An expression evaluating to STRING
- CNUM - An INTEGER literal

Operator precedence

Operator precedence defines the *order of operations* when evaluating an expression. Higher precedence operators are evaluated before those with lower precedence. Operators are listed in the following table as the highest precedence first. For example, `a || b && c` reads as `a || (b && c)` because `&&` has higher precedence than `||`.

Operator *associativity* determines which side binds first for multiple instances of the same operator at equal precedence. For example, `a && b && c` reads as `(a && b) && c` because `&&` has left to right associativity.

Operator	Name	Associativity
`( )`	Grouping for precedence	left to right
`!`	Boolean NOT	right to left
`&&`	Boolean AND	left to right
`||`	Boolean OR	left to right

Negation

Numeric literals may be negated by prefixing the `-` unary operator. This operator may only be applied to literals, and not to numeric values in other contexts.

```
:= [ "-" ] CNUM
 | [ "-" ] CNUM "." [ CNUM ]
```

String concatenation

Adjacent strings are concatenated implicitly, but may also be concatenated explicitly by the `+` operator:

```
:= string-expr string-expr
 | string-expr "+" _string-expr
```

For example, `"abc" "def"` is equivalent to `"abcdef"`.

Assignment and arithmetic operators

The `set` syntax is the only situation in which these operators may be used. Since the operator may only occur once in a `set` statement, these operators are mutually exclusive, so precedence between them is nonsensical.

The values the operators produce are used for assignment only. The `set` statement assigns this value to a variable, but does not itself evaluate to a value.

FLOAT arithmetic has special cases for operands which are NaN: Arithmetic operators evaluate to NaN when either operand is NaN.

FLOAT arithmetic has special cases for operands which are floating point infinities: In general all arithmetic operations evaluate to positive or negative infinity when either operand is infinity. However some situations evaluate to NaN instead. Some of these situations are *domain errors*, in which case `fastly.error` is set to `"EDOM"` accordingly. Others situations are not domain errors: ∞ − ∞ and 0 × ∞. These evaluate to NaN but do not set `fastly.error`.

Assignment

Assignment is provided by the `=` operator:

```
:= "set" variable "=" assignment-expr ";"
```

Addition and subtraction

Addition and subtraction are provided by the `+=` and `-=` operators respectively:

```
:= "set" variable "+=" assignment-expr ";"
 | "set" variable "-=" assignment-expr ";"
```

Multiplication, division and modulus

Multiplication, division and modulus are provided by the `*=`, `/=` and `%=` operators respectively:

```
:= "set" variable "*=" assignment-expr ";"
 | "set" variable "/=" assignment-expr ";"
 | "set" variable "%=" assignment-expr ";"
```

Bitwise operators

```
:= "set" variable "|=" assignment-expr ";"
 | "set" variable "&=" assignment-expr ";"
 | "set" variable "^=" assignment-expr ";"
 | "set" variable ">>=" assignment-expr ";"
 | "set" variable "<<=" assignment-expr ";"
 | "set" variable "ror=" assignment-expr ";"
 | "set" variable "rol=" assignment-expr ";"
```

Right shifts sign-extend negative numbers. For example, `-32 >> 5` gives -1.

Shift and rotate operations with negative shift widths perform the operation in the opposite direction. For example, `32 << -5` gives 1. For right operands larger than the width of `INTEGER`, shifts will yield zero or -1 and rotates will use the operand modulo the width of `INTEGER`.

Logical operators

Logical AND and OR operators are provided by the `&&=` and `||=` operators respectively:

```
:= "set" variable "&&=" assignment-expr ";"
 | "set" variable "||=" assignment-expr ";"
```

These are *short-circuit* operators; see below.

Conditional operators

Conditional operators produce BOOL values, suitable for use in `if` statement conditions.

Logical operators

Conditional expressions may be inverted by prefixing the `!` operator:

```
:= "!" conditional-expr
```

Boolean AND and OR operators (`&&` and `||` respectively) are defined for conditional expressions:

```
:= conditional-expr "&&" conditional-expr
 | conditional-expr "||" conditional-expr
```

These boolean operators have *short-circuit* evaluation, whereby the right-hand operand is only evaluated when necessary in order to compute the resulting value. For example, given `a && b` when the left-hand operand is false, the resulting value will always be false, regardless of the value of the right-hand operand. So in this situation, the right-hand operand will not be evaluated. This can be seen when the right-hand operand has a visible side effect, such as a call to a function which performs some action.

Comparison operators

FLOAT comparisons have special cases for operands which are NaN: The `!=` operator always evaluates to true when either operand is NaN. All other conditional operators always evaluate to false when either operand is NaN. For example, if a given variable is NaN, that variable will compare unequal to itself: both `var.nan == var.nan` and `var.nan >= var.nan` will be false.

STRING comparisons have special cases for operands which are not set (as opposed to empty): The `!=` and `!~` operators always evaluate to true when either operand is not set. All other conditional operators always evaluate to false when either operand is not set. For example, if a

given variable is not set, that variable will compare unequal to itself: both `req.http.unset == req.http.unset` and `req.http.unset ~ ".?"` will be false.

Floating point infinities are signed, and compare as beyond the maximum and minimum values for FLOAT types, such that for any finite value: −∞ < n < +∞

The comparison operators are:

```
lg-op := "<" | ">" | "<=" | ">="
eq-op := "==" | "!="
re-op := "~" | "!~"
```

Equality is defined for all types:

```
:= expr eq-op expr
```

Inequalities are defined for numeric types and TIME:

```
:= numeric-expr lg-op numeric-expr
 | time-expr lg-op time-expr
```

Note that as there are currently no numeric expressions in general; these operators are limited to use with specific operands. For example, `var.i < 5` is permitted but `2 < 5` is not.

Regular expression conditional operators are defined for STRING types and ACLs only:

```
:= string-expr re-op STRING
 | acl re-op STRING
```

The right-hand operand must be a literal string (regular expressions cannot be constructed dynamically).

Reserved punctuation

Punctuation appears in various syntactic roles which are not operators (that is, they do not produce a value).

Punctuation	Example Uses
`{` `}`	Block syntax
`[` `]`	Stats ranges
`(` `)`	Syntax around if conditions, function argument lists
`/`	Netmasks for ACLs
`,`	Separator for function arguments

Punctuation	Example Uses
`;`	Separator for statements and various other syntactic things
`!`	Invert ACL entry
`.`	To prefix fields in backend declarations
`:`	Port numbers for backend declarations, and used in the stats syntax

The following lexical tokens are reserved, but not used: `*` `&` `|` `>>` `<<` `++` `--` `%`

Types (/vcl/types/)

VCL is a statically typed language. Several types are available.

Types for scalar values

These types are provided for scalar values, and may be assigned values from literals. Some types have units; others are unitless.

These types all have implicit conversions to strings, such that their values may be used in contexts where a STRING value is necessary. The rendering for string conversion is not described except for types where it differs from the corresponding literal syntax.

- BOOL (/vcl/types/bool/)
- FLOAT (/vcl/types/float/)
- INTEGER (/vcl/types/integer/)
- IP (/vcl/types/ip/)
- RTIME (/vcl/types/rtime/)
- STRING (/vcl/types/string/)
- TIME (/vcl/types/time/)

Types with special semantics

These types serve as points of abstraction, where internal mechanisms are separated from their interfaces to the VCL syntax. This is either due to special cases for syntax in VCL, or provided for special cases for operations internally.

- BACKEND (/vcl/types/backend/)
- HASH (/vcl/types/hash/)

- HEADER (/vcl/types/header/)
- VOID (/vcl/types/void/)
