VCL

Content negotiation

Content negotiation Functions

accept.charset_lookup()

Selects the best match from a string in the format of an `Accept-Charset` header's value in the listed character sets, using the algorithm described in Section 5.3.3 of RFC 7231.

This function takes the following parameters:

1. a colon-separated list of character sets available for the resource,
2. a fallback return value,
3. a string representing an `Accept-Charset` header's value.

Format

```
STRING
accept.charset_lookup(STRING requested_charsets, STRING default, STRING accept_header)
```

Examples

```
1  set bereq.http.Accept-Charset =
2    accept.charset_lookup("iso-8859-5:iso-8859-2", "utf-8",
3      req.http.Accept-Charset);
```

accept.encoding_lookup()

Selects the best match from a string in the format of an `Accept-Encoding` header's value in the listed content encodings, using the algorithm described in Section 5.3.3 of RFC 7231.

This function takes the following parameters:

1. a colon-separated list of content encodings available for the resource,
2. a fallback return value,
3. a string representing an `Accept-Encoding` header's value.

This function does not have special handling of `x-compress` or `x-gzip` values.

Format

```
STRING
accept.encoding_lookup(STRING requested_content_encodings, STRING default, STRING accept_header)
```

Examples

```
1  set bereq.http.Accept-Encoding =
2    accept.encoding_lookup("compress:gzip", "identity",
3      req.http.Accept-Encoding);
```

accept.language_filter_basic()

Similar to `accept.language_lookup()`, this function selects the best matches from a string in the format of an `Accept-Language` header's value in the listed languages, using the algorithm described in RFC 4647, Section 3.3.1.

This function takes the following parameters:

1. a colon-separated list of languages available for the resource,
2. a fallback return value,
3. a string representing an `Accept-Language` header's value,
4. the maximum number of matching languages to return.

The matches are comma-separated.

Format

```
STRING
accept.language_filter_basic(STRING requested_languages, STRING default, STRING accept_header, INTEGER nmatches)
```

Examples

```
set bereq.http.Accept-Language =
  accept.language_filter_basic("en:de:fr:nl", "nl",
    req.http.Accept-Language, 2);
```

accept.language_lookup()

Selects the best match from a string in the format of an `Accept-Language` header's value in the listed languages, using the algorithm described in RFC 4647, Section 3.4.

This function takes the following parameters:

1. a colon-separated list of languages available for the resource,
2. a fallback return value,
3. a string representing an `Accept-Language` header's value.

This function conforms to RFC 4647.

Format

```
STRING
accept.language_lookup(STRING requested_languages, STRING default, STRING accept_header)
```

Examples

```
set bereq.http.Accept-Language =
  accept.language_lookup("en:de:fr:nl", "en",
    req.http.Accept-Language);
```

accept.media_lookup()

Selects the best match from a string in the format of an `Accept` header's value in the listed media types, using the algorithm described in Section 5.3.2 of RFC 7231.

This function takes the following parameters:

1. a colon-separated list of media types available for the resource,
2. a fallback return value,
3. a colon-separated list of media types, each corresponding to a media type pattern,
4. a string representing an `Accept` header's value.

The matching procedure is case insensitive, matched media types are returned verbatim as supplied to the matching function. Values of the first three arguments can not contain variables. Duplicate media types among the first three arguments are not allowed.

Format

```
STRING
accept.media_lookup(STRING requested_media_types, STRING default, STRING range_defaults, STRING accept_header)
```

Examples

```
# We wish `image/jpeg` to return `image/jpeg`.
# We wish `image/png` to return `image/png`.
# We wish `image/*` to return `image/tiff`.
# We wish `text/*` to return `text/html`.
# We wish `*/*` to return `text/plain`.
set beresp.http.media = accept.media_lookup("image/jpeg:image/png",
  "text/plain",
  "image/tiff:text/html",
  req.http.Accept);
```

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.

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()

Returns an AWSv4 message authentication code 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
digest.awsv4_hmac(STRING key, STRING date_stamp, STRING region, STRING service, STRING string)
```

Examples

```
declare local var.signature STRING;
set var.signature = digest.awsv4_hmac(
    "wJalrXUtnFEMI/K7MDENG+bPxRfiCYEXAMPLEKEY",
    "20120215",
    "us-east-1",
    "iam",
    "hello");
```

digest.base64_decode()

Returns the Base64 decoding of the input string, as specified by RFC 4648.

Format

```
STRING
digest.base64_decode(STRING input)
```

Examples

```
declare local var.base64_decoded STRING;
set var.base64_decoded = digest.base64_decode("zprOsc67z47PgiDOv8+Bzq/Pg86xz4TOtQ==");
# var.base64_decoded is now "Καλώς ορίσατε"
```

digest.base64()

Returns the Base64 encoding of the input string, as specified by RFC 4648.

Format

```
STRING
digest.base64(STRING input)
```

Examples

```
declare local var.base64_encoded STRING;
set var.base64_encoded = digest.base64("Καλώς ορίσατε");
# var.base64_encoded is now "zprOsc67z47PgiDOv8+Bzq/Pg86xz4TOtQ=="
```

digest.base64url_decode()

Returns the Base64 decoding with URL and filename safe alphabet decoding of the input string, as specified by RFC 4648.

Format

```
STRING
digest.base64url_decode(STRING input)
```

Examples

```
declare local var.base64url_decoded STRING;
set var.base64url_decoded = digest.base64url_decode("zprOsc67z47PgiDOv8-Bzq_Pg86xz4TOtQ==");
# var.base64url_decoded is now "Καλώς ορίσατε"
```

digest.base64url_nopad_decode()

Returns the Base64 decoding with URL and filename safe alphabet decoding of the input string, as specified by RFC 4648, without padding (`=`).

Format

```
STRING
digest.base64url_nopad_decode(STRING input)
```

Examples

```
declare local var.base64url_nopad_decoded STRING;
set var.base64url_nopad_decoded = digest.base64url_nopad_decode("zprOsc67z47PgiDOv8-Bzq_Pg86xz4TOtQ");
# var.base64url_nopad_decoded is now "Καλώς ορίσατε"
```

digest.base64url_nopad()

Returns the Base64 encoding with URL and filename safe alphabet encoding of the input string, as specified by RFC 4648, without padding (`=`).

Format

```
STRING
digest.base64url_nopad(STRING input)
```

Examples

```
declare local var.base64url_nopad_encoded STRING;
set var.base64url_nopad_encoded = digest.base64url_nopad("Καλώς ορίσατε");
# var.base64url_nopad_encoded is now "zprOsc67z47PgiDOv8-Bzq_Pg86xz4TOtQ"
```

digest.base64url()

Returns the Base64 encoding with URL and filename safe alphabet of the input string, as specified by RFC 4648.

Format

```
STRING
digest.base64url(STRING input)
```

Examples

```
declare local var.base64url_encoded STRING;
set var.base64url_encoded = digest.base64url("Καλώς ορίσατε");
# var.base64url_encoded is now "zprOsc67z47PgiDOv8-Bzq_Pg86xz4TOtQ=="
```

digest.hash_crc32()

Calculates the 32-bit Cyclic Redundancy Checksum with reversed bit ordering of a string, like that used by bzip2. Returns a hex-encoded string in byte-reversed order, e.g. `181989fc` instead of `fc891918`.

Format

```
STRING
digest.hash_crc32(STRING input)
```

Examples

```
declare local var.crc32 STRING;
set var.crc32 = digest.hash_crc32("123456789");
# var.crc32 is now "181989fc"
```

digest.hash_crc32b()

Calculates the 32-bit Cyclic Redundancy Checksum of a string, as specified by ISO/IEC 13239:2002 and section 8.1.1.6.2 of ITU-T recommendation V.42 and used by Ethernet (IEEE 802.3), V.42, FDDI, gzip, zip, and PNG. Returns a hex-encoded string in byte-reversed order, e.g. `2639f4cb` instead of `cbf43926`.

Format

```
STRING
digest.hash_crc32b(STRING input)
```

Examples

```
declare local var.crc32b STRING;
set var.crc32b = digest.hash_crc32b("123456789");
# var.crc32b is now "2639f4cb"
```

digest.hash_md5()

Use the MD5 hash. Returns a hex-encoded string.

Format

```
STRING
digest.hash_md5(STRING input)
```

Examples

```
declare local var.hash_md5 STRING;
set var.hash_md5 = digest.hash_md5("123456789");
# var.hash_md5 is now "25f9e794323b453885f5181f1b624d0b"
```

digest.hash_sha1()

Use the SHA-1 hash. Returns a hex-encoded string.

Format

```
STRING
digest.hash_sha1(STRING input)
```

Examples

```
declare local var.hash_sha1 STRING;
set var.hash_sha1 = digest.hash_sha1("123456789");
# var.hash_sha1 is now "f7c3bc1d808e04732adf679965ccc34ca7ae3441"
```

digest.hash_sha224()

Use the SHA-224 hash. Returns a hex-encoded string.

Format

```
STRING
digest.hash_sha224(STRING input)
```

Examples

```
declare local var.hash_sha224 STRING;
set var.hash_sha224 = digest.hash_sha224("123456789");
# var.hash_sha224 is now "9b3e61bf29f17c75572fae2e86e17809a4513d07c8a18152acf34521"
```

digest.hash_sha256()

Use the SHA-256 hash. Returns a hex-encoded string.

Format

```
STRING
digest.hash_sha256(STRING input)
```

Examples

```
declare local var.hash_sha256 STRING;
set var.hash_sha256 = digest.hash_sha256("123456789");
# var.hash_sha256 is now "15e2b0d3c33891ebb0f1ef609ec419420c20e320ce94c65fbc8c3312448eb225"
```

digest.hash_sha384()

Use the SHA-384 hash. Returns a hex-encoded string.

Format

```
STRING
digest.hash_sha384(STRING input)
```

Examples

```
declare local var.hash_sha384 STRING;
set var.hash_sha384 = digest.hash_sha384("123456789");
# var.hash_sha384 is now "eb455d56d2c1a69de64e832011f3393d45f3fa31d6842f21af92d2fe469c499da5e3179847334a18479c8d1d
edea1be3"
```

digest.hash_sha512()

Use the SHA-512 hash. Returns a hex-encoded string.

Format

```
STRING
digest.hash_sha512(STRING input)
```

Examples

```
declare local var.hash_sha512 STRING;
set var.hash_sha512 = digest.hash_sha512("123456789");
# var.hash_sha512 is now "d9e6762dd1c8eaf6d61b3c6192fc408d4d6d5f1176d0c29169bc24e71c3f274ad27fcd5811b313d681f7e55ec02d73d499c95455b6b5bb503acf574fba8ffe85"
```

digest.hmac_md5_base64()

Hash-based message authentication code using MD5. Returns a Base64-encoded string.

Format

```
STRING
digest.hmac_md5_base64(STRING key, STRING input)
```

Examples

```
declare local var.hmac_md5_base64 STRING;
set var.hmac_md5_base64 = digest.hmac_md5_base64("key", "input");
# var.hmac_md5_base64 is now "cZ/HW66QBNnoQqSxW4KMBg=="
```

digest.hmac_md5()

Hash-based message authentication code using MD5. Returns a hex-encoded string prepended with 0x.

Format

```
STRING
digest.hmac_md5(STRING key, STRING input)
```

Examples

```
declare local var.hmac_md5 STRING;
set var.hmac_md5 = digest.hmac_md5("key", "input");
# var.hmac_md5 is now "0x719fc75bae9004d9e842a4b15b828c06"
```

digest.hmac_sha1_base64()

Hash-based message authentication code using SHA-1. Returns a Base64-encoded string.

Format

```
STRING
digest.hmac_sha1_base64(STRING key, STRING input)
```

Examples

```
declare local var.hmac_sha1_base64 STRING;
set var.hmac_sha1_base64 = digest.hmac_sha1_base64("key", "input");
# var.hmac_sha1_base64 is now "hRO7NVB2zOKuXrnzmatcr9unyKI="
```

digest.hmac_sha1()

Hash-based message authentication code using SHA-1. Returns a hex-encoded string prepended with 0x.

Format

```
STRING
digest.hmac_sha1(STRING key, STRING input)
```

Examples

```
declare local var.hmac_sha1 STRING;
set var.hmac_sha1 = digest.hmac_sha1("key", "input");
# var.hmac_sha1 is now "0x8513bb355076cce2ae5eb9f399ab5cafdba7c8a2"
```

digest.hmac_sha256_base64()

Hash-based message authentication code using SHA-256. Returns a Base64-encoded string.

Format

```
STRING
digest.hmac_sha256_base64(STRING key, STRING input)
```

Examples

```
declare local var.hmac_sha256_base64 STRING;
set var.hmac_sha256_base64 = digest.hmac_sha256_base64("key", "input");
# var.hmac_sha256_base64 is now "ngiewTr4gaisInpzbD58SQ6jtK/KDF+D3/Y5O2g6cuM="
```

digest.hmac_sha256()

Hash-based message authentication code using SHA-256. Returns a hex-encoded string prepended with 0x.

Format

```
STRING
digest.hmac_sha256(STRING key, STRING input)
```

Examples

```
declare local var.hmac_sha256 STRING;
set var.hmac_sha256 = digest.hmac_sha256("key", "input");
# var.hmac_sha256 is now "0x9e089ec13af881a8ac227a736c3e7c490ea3b4afca0c5f83dff6393b683a72e3"
```

digest.hmac_sha512_base64()

Hash-based message authentication code using SHA-512. Returns a Base64-encoded string.

Format

```
STRING
digest.hmac_sha512_base64(STRING key, STRING input)
```

Examples

```
declare local var.hmac_sha512_base64 STRING;
set var.hmac_sha512_base64 = digest.hmac_sha512_base64("key", "input");
# var.hmac_sha512_base64 is now "A613yBfzJmnMzzjayRXU5VoWgzscpoWXmp31IaBSNZeAkAQ8PaQC2tNl7TmsBa9IZKgERRhh9LTfdoCDTG1PlQ=="
```

digest.hmac_sha512()

Hash-based message authentication code using SHA-512. Returns a hex-encoded string prepended with 0x.

Format

```
STRING
digest.hmac_sha512(STRING key, STRING input)
```

Examples

```
declare local var.hmac_sha512 STRING;
set var.hmac_sha512 = digest.hmac_sha512("key", "input");
# var.hmac_sha512 is now "0x03ad77c817f32669cccf38dac915d4e55a16833b1ca685979a9df521a05235978090043c3da402dad365ed39ac05af4864a804451861f4b4df7680834c6d4f95"
```

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

```
BOOL
digest.rsa_verify(ID hash_method, STRING_LIST public_key, STRING_LIST payload, STRING_LIST digest [, 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()

A boolean function that returns true if s1 and s2 are equal. Comparison time varies on the length of s1 and s2 but not the contents of s1 and s2. For strings of the same length, the comparison is done in constant time to defend against timing attacks.

Format

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

Examples

```
if (!(table.lookup(user2hashedpass, req.http.User) && digest.secure_is_equal(req.http.HashedPass, table.lookup(use
r2hashedpass, req.http.User)))) {
  error 401 "Unauthorized";
}
```

digest.time_hmac_md5()

Returns a time-based one-time password using MD5 based upon the current time. The `key` parameter is a Base64-encoded key. The `interval` parameter specifies the lifetime of the token and must be non-negative. The `offset` parameter provides a means for mitigating clock skew.

Format

```
STRING
digest.time_hmac_md5(STRING key, INTEGER interval, INTEGER offset)
```

Examples

```
set req.http.otp-md5 = digest.time_hmac_md5(digest.base64("secret"), 60, 10);
```

digest.time_hmac_sha1()

Returns a time-based one-time password using SHA-1 based upon the current time. The `key` parameter is a Base64-encoded key. The `interval` parameter specifies the lifetime of the token in seconds and must be non-negative. The `offset` parameter provides a means for mitigating clock skew.

Format

```
STRING
digest.time_hmac_sha1(STRING key, INTEGER interval, INTEGER offset)
```

Examples

```
set req.http.otp-sha-1 = digest.time_hmac_sha1(digest.base64("secret"), 60, 10);
```

digest.time_hmac_sha256()

Returns a time-based one-time password with SHA-256 based upon the current time. The `key` parameter is a Base64-encoded key. The `interval` parameter specifies the lifetime of the token and must be non-negative. The `offset` parameter provides a means for mitigating clock skew.

Format

```
STRING
digest.time_hmac_sha256(STRING key, INTEGER interval, INTEGER offset)
```

Examples

```
set req.http.otp-sha-256 = digest.time_hmac_sha256(digest.base64("secret"), 60, 10);
```

digest.time_hmac_sha512()

Returns a time-based one-time password with SHA-512 based upon the current time. The `key` parameter is a Base64-encoded key. The `interval` parameter specifies the lifetime of the token and must be non-negative. The `offset` parameter provides a means for mitigating clock skew.

Format

```
STRING
digest.time_hmac_sha512(STRING key, INTEGER interval, INTEGER offset)
```

Examples

```
set req.http.otp-sha-512 = digest.time_hmac_sha512(digest.base64("secret"), 60, 10);
```

Date and time

Date and time Functions

parse_time_delta()

Parses a string representing a time delta and returns an integer number of seconds. This function supports the specifiers "d" and "D" for days, "h" and "H" for hours, "m" and "M" for minutes, and "s" and "S" for seconds. The function parses individual deltas like "15m" and "7d". Strings like "10d11h3m2s" are not supported. In case of invalid input, the function returns -1.

Format

```
INTEGER
parse_time_delta(STRING specifier)
```

Examples

```
set beresp.ttl = parse_time_delta(beresp.http.Edge-Control:cache-maxage);
```

std.integer2time()

Converts an integer, representing seconds since the UNIX Epoch, to a time variable.

If the time argument is invalid then this returns a time value which stringifies to: `datetime out of bounds`.

To convert a string, use `std.time()` instead.

Format

```
TIME
std.integer2time(INTEGER time)
```

Examples

```
1  declare local var.once TIME;
2  set var.once = std.integer2time(1136239445);
3  # var.once now represents "Mon, 02 Jan 2006 22:04:05 GMT"
```

std.time()

Converts a string to a time variable.

The following string formats are supported:

- `Mon, 02 Jan 2006 22:04:05 GMT`, RFC 822 and RFC 1123
- `Monday, 02-Jan-06 22:04:05 GMT`, RFC 850
- `Mon Jan 2 22:04:05 2006`, ANSI-C asctime()

- `2006-01-02 22:04:05`, an ISO 8601 subset
- `1136239445.00`, seconds since the UNIX Epoch
- `1136239445`, seconds since the UNIX Epoch

The only time zone supported is `GMT`.

If the string does not match one of those formats, then the fallback variable is returned instead. We recommend using a fallback that's meaningful for your particular Fastly service.

Format

```
TIME
std.time(STRING s, TIME fallback)
```

Examples

```
declare local var.string TIME;
set var.string = std.time("Mon, 02 Jan 2006 22:04:05 GMT", std.integer2time(-1));
# var.string is now "Mon, 02 Jan 2006 22:04:05 GMT"
```

```
declare local var.integer TIME;
set var.integer = std.time("1136239445", std.integer2time(-1));
# var.integer is now "Mon, 02 Jan 2006 22:04:05 GMT"
```

```
declare local var.invalid TIME;
set var.invalid = std.time("Not a date", std.integer2time(-1));
# var.invalid is now "datetime out of bounds"
```

strftime()

Formats a time to a string. This uses standard POSIX strftime formats.

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

```
STRING
strftime(STRING format, TIME time)
```

Examples

```
# Concise format
set resp.http.now = strftime({"%Y-%m-%d %H:%M"}, now);
# resp.http.now is now e.g. 2006-01-02 22:04
```

```
# RFC 5322 format
set resp.http.start = strftime({"%a, %d %b %Y %T %z"}, time.start);
# resp.http.start is now e.g. Mon, 02 Jan 2006 22:04:05 +0000
```

```
# ISO 8601 format
set resp.http.end = strftime({"%Y-%m-%dT%H:%M:%SZ"}, time.end);
# resp.http.end is now e.g. 2006-01-02T22:04:05Z
```

time.add()

Adds a relative time to a time.

Format

```
TIME
time.add(TIME t1, TIME t2)
```

Examples

```
declare local var.one_day_later TIME;
set var.one_day_later = time.add(now, 1d);
# var.one_day_later is now the same time tomorrow
```

time.hex_to_time()

This specialized function takes a hexadecimal string value, divides by `divisor` and interprets the result as seconds since the UNIX Epoch.

Format

```
TIME
time.hex_to_time(INTEGER divisor, STRING dividend)
```

Examples

```
declare local var.hextime TIME;
set var.hextime = time.hex_to_time(1, "43b9a355");
# var.hextime is now "Mon, 02 Jan 2006 22:04:05 GMT"
```

time.is_after()

Returns true if `t1` is after `t2`. (Normal timeflow and causality required.)

Format

```
BOOL
time.is_after(TIME t1, TIME t2)
```

Examples

```
if (time.is_after(time.add(now, 10m), now)) {
  ...
}
```

time.sub()

Subtracts a relative time from a time.

Format

```
TIME
time.sub(TIME t1, TIME t2)
```

Examples

```
declare local var.one_day_earlier TIME;
set var.one_day_earlier = time.sub(now, 1d);
# var.one_day_earlier is now the same time yesterday
```

Date and time Variables

now.sec

Like the `now` variable, but in seconds since the UNIX Epoch.

Type

`STRING`

Accessibility

Readable From

All subroutines

now

The current time in RFC 1123 format (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`).

Type

`TIME`

Accessibility

Readable From

All subroutines

time.elapsed.msec_frac

The time that has elapsed in milliseconds since the request started.

Type

`STRING`

Accessibility

Readable From

All subroutines

time.elapsed.msec

The time since the request start in milliseconds.

Type

`STRING`

Accessibility

Readable From

All subroutines

time.elapsed.sec

The time since the request start in seconds.

Type

`STRING`

Accessibility

Readable From

All subroutines

time.elapsed.usec_frac

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

Type

`STRING`

Accessibility

Readable From

All subroutines

time.elapsed.usec

The time since the request start in microseconds.

Type

`STRING`

Accessibility

Readable From

All subroutines

time.elapsed

The time since the request started. Also useful for `strftime()`.

Type

`RTIME`

Accessibility

Readable From

All subroutines

time.end.msec_frac

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

Type

`STRING`

Accessibility

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.msec

The time the request ended in milliseconds since the UNIX Epoch.

Type

`STRING`

Accessibility

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.sec

The time the request ended in seconds since the UNIX Epoch.

Type

`STRING`

Accessibility

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.usec_frac

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

Type

`STRING`

Accessibility

Readable From

- `vcl_deliver`
- `vcl_log`

time.end.usec

The time the request ended in microseconds since the UNIX Epoch.

Type

`STRING`

Accessibility

Readable From

- `vcl_deliver`
- `vcl_log`

time.end

The time the request ended, using RFC 1123 format (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`). Also useful for `strftime()`.

Type

`TIME`

Accessibility

Readable From

All subroutines

time.start.msec_frac

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

Type

`STRING`

Accessibility

Readable From

All subroutines

time.start.msec

The time the request started in milliseconds since the UNIX Epoch, after TLS termination.

Type

`STRING`

Accessibility

Readable From

All subroutines

time.start.sec

The time the request started in seconds since the UNIX Epoch, after TLS termination.

Type

`STRING`

Accessibility

Readable From

All subroutines

time.start.usec_frac

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

Type

`STRING`

Accessibility

Readable From

All subroutines

time.start.usec

The time the request started in microseconds since the UNIX Epoch, after TLS termination.

Type

`STRING`

Accessibility

Readable From

All subroutines

time.start

The time the request started, after TLS termination, using RFC 1123 format (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`).

Type

`TIME`

Accessibility

Readable From

All subroutines

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.

Type

`RTIME`

Accessibility

Readable From

- `vcl_deliver`
- `vcl_log`

Edge Side Includes (ESI)

Edge Side Includes (ESI) Variables

req.esi

Whether or not to disable or enable ESI processing during this request. Using `set req.esi = false;` will disable ESI processing. The default value is `true`.

Type

`BOOL`

Accessibility

Readable From

- `vcl_recv`
- `vcl_fetch`
- `vcl_deliver`
- `vcl_error`

req.topurl

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

Type

`STRING`

Accessibility

Readable From

All subroutines

Floating point classification

Floating point values are grouped into one of several *classifications*:

- **Finite** — math.is_finite()

  A value that is neither NaN nor an infinity.

- **Subnormal** — math.is_subnormal()

  The `FLOAT` type supports subnormals (also known as *denormals*).

- **NaN** — math.is_nan()

  The `FLOAT` type may express NaN (Not a Number). In general, arithmetic operations involving a NaN will produce NaN. NaN values are signaled through the `fastly.error` variable.

  There is no literal syntax for assigning NaN, but a math.NAN constant is provided.

- **Normal** — math.is_normal()

  A value that is neither NaN, subnormal, an infinity nor a zero.

  Note that "normal" is not the exact opposite of "subnormal" because of the other possible non-subnormal values.

- **Infinite** — math.is_infinite()

  The `FLOAT` type may express IEEE 754 *infinities*. These are signed values. Infinities behave with special semantics for some operators.

  There is no literal syntax for assigning infinities, but math.POS_INFINITY and math.NEG_INFINITY constants are provided.

- **Zero** — There are two kinds of zero: positive and negative. Both compare equal.

  No VCL function is provided to determine whether a floating point value is a zero. Because both positive and negative zero compare equal, a comparison may be made simply by `var.x == 0`.

Floating point classification Functions

math.is_finite()

Determines whether a floating point value is finite. See floating point classifications for more information.

Format

```
BOOL
math.is_finite(FLOAT x)
```

math.is_infinite()

Determines whether a floating point value is an infinity. See floating point classifications for more information.

Format

```
BOOL
math.is_infinite(FLOAT x)
```

Examples

```
declare local var.f FLOAT;

set var.f = math.POS_INFINITY;
set var.f -= 1; # +∞ - 1 produces +∞
if (math.is_infinite(var.f)) {
  log "infinity";
}
```

math.is_nan()

Determines whether a floating point value is NaN (Not a Number). See floating point classifications for more information.

Format

```
BOOL
math.is_nan(FLOAT x)
```

Examples

```
declare local var.f FLOAT;

set var.f = 1;
set var.f /= 0;
if (math.is_nan(var.f)) {
  log "division by zero";
}
```

math.is_normal()

Determines whether a floating point value is normal. See floating point classifications for more information.

Format

```
BOOL
math.is_normal(FLOAT x)
```

Examples

```
# zeroes are not normals
if (!math.is_normal(0)) {
  log "not a normal";
}
```

math.is_subnormal()

Determines whether a floating point value is subnormal. See floating point classifications for more information.

Format

```
BOOL
math.is_subnormal(FLOAT x)
```

Examples

```
declare local var.f FLOAT;

set var.f = math.FLOAT_MIN; # minimum finite value
if (!math.is_subnormal(var.f)) {
  log "not subnormal";
}
set var.f /= 2;
if (math.is_subnormal(var.f)) {
  log "subnormal";
}
```

Geolocation

All geographic data presented through these variables is associated with a particular IP address. This address is automatically populated from `client.ip` by default, but may be overridden explicitly by setting `client.geo.ip_override`.

Geographic variables representing names are available in several encodings. Note in particular the `*.ascii` variables are lossy. These variables have diacritics removed and are normalized to lowercase spellings. These `*.ascii` variables can be used as a symbolic string in code (for example, to perform some different action depending on the city name). Due to their simplified content, however, they are generally inappropriate for presenting to users.

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, 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 an IP geolocation dataset vendor, such as Digital Element.

TIP: If you're updating your configurations from older version of the geolocation variables, be sure to read our migration guide.

Using geographic variables with shielding

If you have shielding enabled, you should set the following variable before using geographic variables:

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

Absent data

WARNING: The geolocation data is updated periodically as IP allocations change and various amendments are made. Some variables may be absent for the current data at any given time.

For `STRING` types, the special value `?` is used to indicate absent data. These may be normalized to VCL empty strings using the `if()` ternary operator:

```
log if (client.as.name == "?", client.as.name, "");
```

In general strings in VCL may be *not set* (see the VCL documentation for types). This never occurs for the geolocation variables.

Reserved IP address blocks

The IPv4 and IPv6 address spaces have several blocks reserved for special uses. These include private use networks (e.g., 192.168.0.0/16), loopback (127.0.0.1/8), and address ranges reserved for documentation (e.g., 203.0.113.0/24 RFC 5737 TEST-NET-3).

Geographic data has no meaningful association for these ranges. The geolocation VCL variables present special values for these ranges instead. These values are:

Variable	Value for reserved blocks
`client.as.number`	0
`client.as.name`	`?`
`client.geo.latitude`	`0.000`
`client.geo.longitude`	`0.000`
`client.geo.conn_speed`	`broadband`
`client.geo.metro_code`	-1
`client.geo.gmt_offset`	9999
`client.geo.area_code`	0

Variable	Value for reserved blocks
`client.geo.postal_code`	0
`client.geo.continent_code`	`**`
`client.geo.country_code`	`**`
`client.geo.country_code3`	`***`
`client.geo.country_name`	`reserved/private`
`client.geo.city`	`reserved`
`client.geo.region`	`***`

Geolocation Variables

client.as.name

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

For example, `fastly` is the value given for IP addresses under AS-54113.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.as.number

Autonomous system (AS) number.

The `INTEGER` type in VCL is wide enough to support the full range of 32-bit AS numbers.

Formatting these numbers to base 10 (e.g., by implicit conversion to a `STRING` type) will give an `asplain` representation of the number, which is just its base 10 representation.

RFC 5396 introduces the `asdot+` format, which represents a 32-bit AS number as two 16-bit parts. The following VCL illustrates constructing an `asdot+` formatted number:

```
1  declare local var.hi INTEGER;
2  declare local var.lo INTEGER;
3  set var.hi = client.as.number;
4  set var.hi >>= 16;
5  set var.lo = client.as.number;
6  set var.lo &= 0xFFFF;
7  log client.as.number ": " var.hi "." var.lo;
```

Examples

The 32-bit AS number 65550 (reserved by RFC 5398 for documentation use) is rendered as `1.14`.

Several ranges of AS numbers are reserved for various purposes. The following VCL fragment illustrates categorizing AS numbers into these ranges:

```
declare local var.as_category STRING;
if (client.as.number < 0 || client.as.number > 0xFFFFFFFF) {
  set var.as_category = "invalid";
} else if (client.as.number == 0) {
  set var.as_category = "reserved"; # RFC 1930
} else if (client.as.number <= 23455) {
  set var.as_category = "public";
} else if (client.as.number == 23456) {
  set var.as_category = "transition"; # RFC 6793
} else if (client.as.number <= 64534) {
  set var.as_category = "public";
} else if (client.as.number <= 64495) {
  set var.as_category = "reserved";
} else if (client.as.number <= 64511) {
  set var.as_category = "documentation"; # RFC 5398
} else if (client.as.number <= 65534) {
  set var.as_category = "private";
} else if (client.as.number == 65535) {
  set var.as_category = "reserved";
} else if (client.as.number <= 65551) {
  set var.as_category = "documentation"; # RFC 4893, RFC 5398
} else if (client.as.number <= 131071) {
  set var.as_category = "reserved";
} else if (client.as.number <= 4199999999) {
  set var.as_category = "public";
} else if (client.as.number <= 4294967294) {
  set var.as_category = "private"; # RFC 6996
} else if (client.as.number == 4294967295) {
  set var.as_category = "reserved";
} else {
  set var.as_category = "unknown";
}
```

Type

INTEGER

Accessibility

Readable From

All subroutines

client.geo.area_code

The telephone area code associated with the IP address. These are only available for IP addresses in the United States, its territories, and Canada.

Type

INTEGER

Accessibility

Readable From

All subroutines

client.geo.city.ascii

City or town name, encoded using ASCII encoding. Lowercase ASCII approximation of the `.utf8` string with diacritics removed.

Type

STRING

Accessibility

Readable From

All subroutines

client.geo.city.latin1

City or town name, encoded using Latin-1 encoding.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.city.utf8

City or town name, encoded using UTF-8 encoding.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.city

Alias of `client.geo.city.ascii`.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.conn_speed

Connection speed. These connection speeds imply different latencies, as well as throughput.

Possible values are: `broadband`, `cable`, `dialup`, `mobile`, `oc12`, `oc3`, `t1`, `t3`, `satellite`, `wireless`, `xdsl`.

See OC rates and T-carrier for background on OC- and T- connections.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.continent_code

Two-letter code representing the continent. Possible codes are:

Code	Continent
`AF`	Africa
`AN`	Antarctica
`AS`	Asia
`EU`	Europe
`NA`	North America
`OC`	Oceania
`SA`	South America

These continents are defined by UN M.49.

The continent code for the Caribbean countries is `NA`.

Note that `EU` refers to the continent name, not to the European Union. For example, IP addresses allocated to Norway and Switzerland (members of the European Economic Area and the Schengen Area respectively, but not of the European Union) are presented with the continent code EU, meaning the geographic continent of Europe.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.country_code

A two-character 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.

These values include subdivisions that are assigned their own country codes in ISO 3166-1. For example, subdivisions NO-21 and NO-22 are presented with the country code SJ for Svalbard and the Jan Mayen Islands.

Examples

The following VCL fragment uses a two-letter country code to construct an emoji flag from its corresponding Unicode regional indicator symbols:

```
table unicode_ri {
  "A": "%u{1F1E6}", "B": "%u{1F1E7}", "C": "%u{1F1E8}", "D": "%u{1F1E9}",
  "E": "%u{1F1EA}", "F": "%u{1F1EB}", "G": "%u{1F1EC}", "H": "%u{1F1ED}",
  "I": "%u{1F1EE}", "J": "%u{1F1EF}", "K": "%u{1F1F0}", "L": "%u{1F1F1}",
  "M": "%u{1F1F2}", "N": "%u{1F1F3}", "O": "%u{1F1F4}", "P": "%u{1F1F5}",
  "Q": "%u{1F1F6}", "R": "%u{1F1F7}", "S": "%u{1F1F8}", "T": "%u{1F1F9}",
  "U": "%u{1F1FA}", "V": "%u{1F1FB}", "W": "%u{1F1FC}", "X": "%u{1F1FD}",
  "Y": "%u{1F1FE}", "Z": "%u{1F1FF}"
}

set resp.http.X-flag = table.lookup(unicode_ri, substr(client.geo.country_code, 0, 1))
                       table.lookup(unicode_ri, substr(client.geo.country_code, 1, 1));
```

For example, the country code SE will produce 🇸🇪 (the Swedish flag).

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.country_code3

A three-character 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.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.country_name.ascii

Country name, encoded using ASCII encoding.

This field is a lowercase transliteration of the ISO 3166-1 English short name for a country.

Examples

For example, the English short name for `FK` is `FALKLAND ISLANDS (MALVINAS)` and so the corresponding value of `client.geo.country_name.ascii` is `falkland islands (malvinas)` (e.g., converted to lowercase).

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.country_name.latin1

Country name, encoded using Latin-1 encoding.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.country_name.utf8

Country name, encoded using UTF-8 encoding.

This field is the ISO 3166-1 English short name for a country.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.country_name

Alias of `client.geo.country_name.ascii`.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.gmt_offset

Time zone offset from coordinated universal time (UTC) for `client.geo.city`.

> **NOTE:** Despite its name, this is not the offset from GMT.

Values may be negative. Values are given as base-10 numbers of three or four digits in the form `(-)HHMM` or `(-)HMM` where `H` is hours and `M` is minutes. For example, -230 would be offset of minus two hours and thirty minutes from UTC.

This may be formatted to an ISO 8601 four-digit form `(-)HHMM` using VCL:

```
1  declare local var.offset STRING;
2  set var.offset = regsub(client.geo.gmt_offset, "^(-?)(...)$", "\10\2");
```

The special value 0 is used to indicate absent data. The special value 9999 is used to indicate an invalid region.

Examples

Not all timezone offsets are on the hour. For example, in St. John's, Newfoundland, `client.geo.gmt_offset` may be -230 or -330 (depending on daylight savings time). The following VCL fragment produces a value in units of hours:

```
declare local var.offset_by_hour FLOAT;
set var.offset_by_hour = client.geo.gmt_offset;
set var.offset_by_hour %= 100;
set var.offset_by_hour /= 60; # minutes
set var.offset_by_hour += std.atoi(regsub(client.geo.gmt_offset, "..$", "")); # truncate
```

Here, increments of 0.5 correspond to half hours. For example, an offset of 930 will produce a floating point value of 9.5.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

client.geo.ip_override

Override the IP address for geolocation data. The default is to use geolocation data for `client.ip`.

It is possible to set `client.geo.ip_override` to an invalid IP address:

```
set client.geo.ip_override = "xxx";
```

in which case the various geolocation variables present values to indicate an invalid region. `STRING` variables are set to the empty string, `FLOAT` variables are set to 999.0, and `INTEGER` variables are set to 0.

Type

`IP`

Accessibility

Readable From

All subroutines

client.geo.latitude

Latitude, in units of degrees from the equator. Values range from -90 to +90 inclusive, with the exception of the special value 999.9 used to indicate absent data.

The latitude given is based on the WGS 84 coordinate reference system.

Examples

An example showing construction of a geo URI as specified by RFC 5870 in VCL:

```
declare local var.geouri STRING;
set var.geouri = "geo:" + client.geo.latitude + "," + client.geo.longitude;
```

This produces a URI of the form `geo:37.786971,-122.399677` (where WGS 84 is the default CRS).

Here's an example showing classification to the five main geographical zones in VCL (latitude values as of October 2018):

```
declare local var.zone STRING;
if (client.geo.latitude == 999.9) {
  set var.zone = "";
} else if (client.geo.latitude >= 66.5) { # Arctic circle
  set var.zone = "North frigid";
} else if (client.geo.latitude >= 23.5) { # Topic of Cancer
  set var.zone = "North temperate";
} else if (client.geo.latitude <= -66.5) { # Antarctic Circle
  set var.zone = "South frigid";
} else if (client.geo.latitude <= -23.5) { # Tropic of Capricorn
  set var.zone = "South temperate";
} else {
  set var.zone = "Torrid";
}
```

You can use VCL to convert to degrees, minutes and seconds:

```
declare local var.deg INTEGER;
declare local var.min INTEGER;
declare local var.sec FLOAT;

declare local var.angle FLOAT;
declare local var.whole FLOAT;
declare local var.frac FLOAT;

set var.angle = client.geo.latitude; # input
if (var.angle < 0.0) {
  set var.angle *= -1;
}

set var.frac = var.angle;
set var.whole = var.frac;
set var.frac %= 1.0;
set var.whole -= var.frac;
set var.deg = var.whole; # truncated, integer by rounding

set var.frac *= 60.0;
set var.whole = var.frac;
set var.frac %= 1.0;
set var.whole -= var.frac;
set var.min = var.whole; # truncated, integer by rounding

set var.sec = var.frac;
set var.sec *= 60.0; # floating seconds

log client.geo.latitude + " = " + var.deg "° " var.min "′ " var.sec "″ "
  + if (client.geo.latitude < 0.0, "S", "N");
```

For example, a latitude of 59.926 produces `59° 55′ 33.600″ N`. The `′` and `″` symbols are Unicode prime symbols, not quotes.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

client.geo.longitude

Longitude, in units of degrees from the IERS Reference Meridian. Values range from -180 to +180 inclusive, with the exception of the special value 999.9 used to indicate absent data.

The longitude given is based on the WGS 84 coordinate reference system.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

client.geo.metro_code

Metro code.

Metro codes represent designated market areas (DMAs) in the United States.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

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. For countries with alphanumeric postal codes, this field is a lowercase transliteration.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.region.ascii

ISO 3166-2 country subdivision code. For countries with multiple levels of subdivision (for example, nations within the United Kingdom), this variable gives the more specific subdivision.

The special value `NO REGION` is given for countries that do not have ISO country subdivision codes. For example, `NO REGION` is given for IP addresses assigned to the Åland Islands (country code AX, illustrated below).

These region values are the subdivision part only. For typical use, a subdivision is normally formatted with its associated country code. The following VCL fragment illustrates constructing an ISO 3166-2 two-part country and subdivision code from the respective variables:

```
declare local var.code STRING;
if (client.geo.country_code != "**") {
  set var.code = client.geo.country_code;
  if (client.geo.region != "NO REGION" && client.geo.region != "?") {
    set var.code = var.code + "-" + client.geo.region;
  }
}
```

Examples

Here are some example values:

`var.code`	**Region Name**	**Country**	**ISO 3166-2 subdivision**
`AX`	Ödkarby	Åland Islands	(none)
`DE-BE`	Berlin	Germany	Land (State)
`GB-BNH`	Brighton and Hove	United Kingdom	Unitary authority
`JP-13`	東京都 (Tōkyō-to)	Japan	Prefecture
`RU-MOW`	Москва́ (Moscow)	Russian Federation	Federal city
`SE-AB`	Stockholms län	Sweden	Län (County)
`US-CA`	California	United States	State

Here, the region name is given for sake of reference only. The region name is not provided as a VCL variable.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.region.latin1

Region code, encoded using Latin-1 encoding.

Because this is a code and contains alphanumeric Latin characters only, it will always be identical to `client.geo.region.ascii`.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.region.utf8

Region code, encoded using UTF-8 encoding.

Because this is a code and contains alphanumeric Latin characters only, it will always be identical to `client.geo.region.ascii`.

Type

`STRING`

Accessibility

Readable From

All subroutines

client.geo.region

Alias of `client.geo.region.ascii`.

Type

`STRING`

Accessibility

Readable From

All subroutines

Math constants and limits

Math constants and limits Variables

math.1_PI

The value of the reciprocal of `math.PI` (1/Pi).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.2_PI

The value of two times the reciprocal of `math.PI` (2/Pi).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.2_SQRTPI

The value of two times the reciprocal of the square root of `math.PI` (2/sqrt(Pi)).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.2PI

The value of `math.PI` multiplied by two (Tau).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.E

The value of the base of natural logarithms (e).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.FLOAT_DIG

Number of decimal digits that can be stored without loss in the `FLOAT` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.FLOAT_EPSILON

Minimum positive difference from 1.0 for the `FLOAT` type.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.FLOAT_MANT_DIG

Number of hexadecimal digits stored for the significand in the `FLOAT` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.FLOAT_MAX_10_EXP

Maximum value in base 10 of the exponent part of the `FLOAT` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.FLOAT_MAX_EXP

Maximum value in base 2 of the exponent part of the `FLOAT` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.FLOAT_MAX

Maximum finite value for the `FLOAT` type.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.FLOAT_MIN_10_EXP

Minimum value in base 10 of the exponent part of the `FLOAT` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.FLOAT_MIN_EXP

Minimum value in base 2 of the exponent part of the `FLOAT` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.FLOAT_MIN

Minimum finite value for the `FLOAT` type.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.INTEGER_BIT

Number of bits in the `INTEGER` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.INTEGER_MAX

Maximum value for the `INTEGER` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.INTEGER_MIN

Minimum value for the `INTEGER` type.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

math.LN10

The value of the natural logarithm of 10 (log_e 10).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.LN2

The value of the natural logarithm of 2 (log_e 2).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.LOG10E

The value of the logarithm to base 10 of `math.E` (log_10 e).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.LOG2E

The value of the logarithm to base 2 of `math.E` (log_2 e).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.NAN

A value that is "not a number." When converted to a STRING value, this is rendered as `NaN`.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.NEG_HUGE_VAL

Negative overflow value.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.NEG_INFINITY

A value representing negative infinity (−∞). When converted to a STRING value, this is rendered as `-inf`.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.PHI

The golden ratio (Φ).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.PI_2

The value of `math.PI` divided by two (Pi/2).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.PI_4

The value of `math.PI` divided by four (Pi/4).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.PI

The value of the ratio of a circle's circumference to its diameter (Pi).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.POS_HUGE_VAL

Positive overflow value.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.POS_INFINITY

A value representing positive infinity (+∞). When converted to a STRING value, this is rendered as `inf`.

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.SQRT1_2

The value of the reciprocal of the square root of two (1/sqrt(2)).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.SQRT2

The value of the square root of two (sqrt(2)).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

math.TAU

The value of `math.PI` multiplied by two (Tau).

Type

`FLOAT`

Accessibility

Readable From

All subroutines

Math rounding

See rounding modes for details of the rounding modes provided by these functions and for an overview of example values.

Math rounding Functions

math.ceil()

Computes the smallest integer value greater than or equal to the given value. In other words, round *x* towards positive infinity.

For example, 2.2, 2.5, and 2.7 all ceil to 3.0.

Return Value

If *x* is `math.NAN`, a NaN will be returned.

If *x* is integral, ±0, *x* itself is returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, an infinity of the same sign is returned.

Otherwise, the rounded value of *x* is returned.

Format

```
FLOAT
math.ceil(FLOAT x)
```

math.floor()

Computes the largest integer value less than or equal to the given value. In other words, round *x* towards negative infinity.

For example, 2.2, 2.5, and 2.7 all floor to 2.0.

Return value

If *x* is `math.NAN`, a NaN will be returned.

If *x* is integral, ±0, *x* itself is returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, an infinity of the same sign is returned.

Otherwise, the rounded value of *x* is returned.

Format

```
FLOAT
math.floor(FLOAT x)
```

math.round()

Rounds *x* to the nearest integer, with ties away from zero (*commercial rounding*).

Return value

If *x* is `math.NAN`, a NaN will be returned.

If *x* is integral, ±0, *x* itself is returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, an infinity of the same sign is returned.

Otherwise, the rounded value of *x* is returned.

Format

```
FLOAT
math.round(FLOAT x)
```

math.roundeven()

Rounds *x* to nearest, ties to even (*bankers' rounding*).

Return value

If *x* is `math.NAN`, a NaN will be returned.

If *x* is integral, ±0, *x* itself is returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, an infinity of the same sign is returned.

Otherwise, the rounded value of *x* is returned.

Format

```
FLOAT
math.roundeven(FLOAT x)
```

math.roundhalfdown()

Rounds to nearest, ties towards negative infinity (*half down*).

Return value

If *x* is `math.NAN`, a NaN will be returned.

If *x* is integral, ±0, *x* itself is returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, an infinity of the same sign is returned.

Otherwise, the rounded value of *x* is returned.

Format

```
FLOAT
math.roundhalfdown(FLOAT x)
```

math.roundhalfup()

Rounds to nearest, ties towards positive infinity (*half up*).

Return value

If *x* is `math.NAN`, a NaN will be returned.

If *x* is integral, ±0, *x* itself is returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, an infinity of the same sign is returned.

Otherwise, the rounded value of *x* is returned.

Format

```
FLOAT
math.roundhalfup(FLOAT x)
```

math.trunc()

Truncates *x* to an integer value less than or equal in absolute value. In other words, rounds *x* towards zero. Negative values will be rounded up towards zero and positive values will be rounded down towards zero.

For example, 2.2, 2.5, and 2.7 all truncate to 2.0.

This is equivalent to formatting the number to base ten and removing all digits after the decimal point.

Return value

If *x* is `math.NAN`, a NaN will be returned.

If *x* is integral, ±0, *x* itself is returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, an infinity of the same sign is returned.

Otherwise, the rounded value of *x* is returned.

Format

```
FLOAT
math.trunc(FLOAT x)
```

Math trigonometric

Math trigonometric Functions

math.acos()

Computes the principal value of the arc cosine of its argument *x*.

Parameters

x - Floating point value. The value of *x* should be in the range -1 to 1 inclusive.

Return value

Upon successful completion, this function returns the arc cosine of *x* in the range 0 to `math.PI` radians inclusive.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is +1, +0 will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, a domain error occurs and a NaN will be returned.

For finite values of x not in the range -1 to 1 inclusive, a domain error occurs and a NaN will be returned.

Errors

If the *x* argument is finite and is not in the range -1 to 1 inclusive, or is `math.POS_INFINITY` or `math.NEG_INFINITY`, then `fastly.error` will be set to `EDOM`.

Format

```
FLOAT
math.acos(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.cos(1.1); // Returns math.NAN
4
5  if (faslty.error) {
6    set resp.http.acos-error = faslty.error; // Returns "EDOM"
7  }
```

math.acosh()

Computes the inverse hyperbolic cosine of its argument *x*.

Parameters

x - Floating point value representing the area of a hyperbolic sector.

Return value

Upon successful completion, this function returns the inverse hyperbolic cosine of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is +1, +0 will be returned.

If *x* is `math.POS_INFINITY`, `math.POS_INFINITY` will be returned.

If *x* is `math.NEG_INFINITY`, a domain error occurs and a NaN will be returned.

For finite values of *x* < 1, a domain error occurs and a NaN will be returned.

Errors

If the *x* argument is finite and less than +1.0, or is `math.NEG_INFINITY`, then `fastly.error` will be set to `EDOM`.

Format

```
FLOAT
math.acosh(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.acosh(10);
```

math.asin()

Computes the principal value of the arc sine of the argument *x*.

Parameters

x - Floating point value. The value of *x* should be in the range -1 to 1 inclusive.

Return value

Upon successful completion, this function returns the arc sine of *x*, in the range -`math.PI_2` to `math.PI_2` radians inclusive.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, *x* will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, a domain error occurs and a NaN will be returned.

If *x* is subnormal, a range error occurs and *x* will be returned.

For finite values of x not in the range -1 to 1 inclusive, a domain error occurs and a NaN will be returned.

Errors

- If the *x* argument is finite and is not in the range -1 to 1 inclusive, or is `math.POS_INFINITY` or `math.NEG_INFINITY`, then `fastly.error` will be set to `EDOM`.
- If the *x* argument is subnormal, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.asin(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.asin(1.0);
```

math.asinh()

Computes the inverse hyperbolic sine of its argument *x*.

Parameters

x - Floating point value representing the area of a hyperbolic sector.

Return value

Upon successful completion, this function returns the inverse hyperbolic sine of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, or `math.POS_INFINITY` or `math.NEG_INFINITY`, *x* will be returned.

If *x* is subnormal, a range error occurs and *x* will be returned.

Errors

If the *x* argument is subnormal, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.asinh(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.asinh(1);
```

math.atan()

Computes the principal value of the arc tangent of its argument *x*.

Parameters

x - Floating point value.

Return value

Upon successful completion, this function returns the arc tangent of *x* in the range -`math.PI_2` to `math.PI_2` radians inclusive.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, *x* will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, ±`math.PI_2` will be returned.

If *x* is subnormal, a range error occurs and *x* will be returned.

Errors

If the *x* argument is subnormal, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.atan(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.atan(1);
```

math.atan2()

Computes the principal value of the arc tangent of *y/x*, using the signs of both arguments to determine the quadrant of the Return Value.

Parameters

- *y* - Floating point value.
- *x* - Floating point value.

Return value

Upon successful completion, this function returns the arc tangent of *y/x* in the range -`math.PI` to `math.PI` radians inclusive.

If *y* is ±0 and *x* is < 0, ±`math.PI` will be returned.

If *y* is ±0 and *x* is > 0, ±0 will be returned.

If *y* is < 0 and *x* is ±0, -`math.PI_2` will be returned.

If *y* is > 0 and *x* is ±0, `math.PI_2` will be returned.

If *x* is 0, a pole error will not occur.

If either *x* or *y* is `math.NAN`, a NaN will be returned.

If *y* is ±0 and *x* is +0, ±0 will be returned.

For finite values of ±*y* > 0, if *x* is `math.NEG_INFINITY`, ±`math.PI` will be returned.

For finite values of ±*y* > 0, if *x* is `math.POS_INFINITY`, ±0 will be returned.

For finite values of *x*, if *y* is `math.POS_INFINITY` or `math.NEG_INFINITY`, ±`math.PI_2` will be returned.

If *y* is `math.POS_INFINITY` or `math.NEG_INFINITY` and *x* is `math.NEG_INFINITY`, ±(3*`math.PI_4`) will be returned.

If *y* is `math.POS_INFINITY` or `math.NEG_INFINITY` and *x* is `math.POS_INFINITY`, ±`math.PI_4` will be returned.

If both arguments are 0, a domain error will not occur.

If the result would cause an underflow, a range error occurs, and `math.atan2()` will return *y/x*.

Errors

No errors occur.

Format

```
FLOAT
math.atan2(FLOAT y, FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.atan2(7, -0);
```

math.atanh()

Computes the inverse hyperbolic tangent of its argument *x*.

Parameters

x - Floating point value representing a hyperbolic angle.

Return value

Upon successful completion, this function returns the inverse hyperbolic tangent of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, *x* will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, a domain error occurs and a NaN will be returned.

If *x* is subnormal, a range error occurs and *x* will be returned.

For finite |*x*|>1, a domain error occurs and a NaN will be returned.

If *x* is ±1, a pole error occurs, and `math.atanh()` will return the value of the macro `math.POS_HUGE_VAL` or `math.NEG_HUGE_VAL` with the same sign as the result of the function.

Errors

- If the *x* argument is finite and not in the range -1 to 1 inclusive, or if it is `math.POS_INFINITY` or `math.NEG_INFINITY`, then `fastly.error` will be set to `EDOM`.
- If the *x* argument is subnormal, or ±1, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.atanh(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.atanh(-1); // Returns math.NEG_INFINITY
4
5  if (fastly.error) {
6    set resp.http.atanh-error = faslty.error; // Returns "ERANGE"
7  }
```

math.cos()

Computes the cosine of its argument *x*, measured in radians.

Parameters

x - Floating point value representing an angle in radians.

Return value

Upon successful completion, this function returns the cosine of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, the value 1.0 will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, a domain error occurs and a NaN will be returned.

Errors

- If the *x* argument is `math.POS_INFINITY` or `math.NEG_INFINITY`, then `fastly.error` will be set to `EDOM`.

Format

```
FLOAT
math.cos(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.cos(math.PI_2);
```

math.cosh()

Computes the hyperbolic cosine of its argument *x*.

Parameters

x - Floating point value representing a hyperbolic angle.

Return value

Upon successful completion, this function returns the hyperbolic cosine of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, the value 1.0 will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, `math.POS_INFINITY` will be returned.

If the result would cause an overflow, a range error occurs and `math.cosh()` will return the value of the macro `math.POS_HUGE_VAL`.

Errors

If the result would cause an overflow, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.cosh(FLOAT x)
```

Examples

```
declare local var.fo FLOAT;

set var.fo = math.cosh(0);
```

math.sin()

Computes the sine of its argument *x*, measured in radians.

Parameters

x - Floating point value representing an angle in radians.

Return value

Upon successful completion, this function returns the sine of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, *x* will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, a domain error occurs and a NaN will be returned.

If *x* is subnormal, a range error occurs and *x* will be returned.

Errors

- If the *x* argument is `math.POS_INFINITY` or `math.NEG_INFINITY`, then `fastly.error` will be set to `EDOM`.
- If the *x* argument is subnormal, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.sin(FLOAT x)
```

Examples

```
declare local var.fi FLOAT;
declare local var.fo FLOAT;

set var.fi = math.PI;
set var.fi /= 6;
set var.fo = math.sin(var.fi);
```

math.sinh()

Computes the hyperbolic sine of its argument *x*.

Parameters

x - Floating point value representing a hyperbolic angle.

Return value

Upon successful completion, this function returns the hyperbolic sine of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, or `math.POS_INFINITY` or `math.NEG_INFINITY`, *x* will be returned.

If *x* is subnormal, a range error occurs and *x* will be returned.

If the result would cause an overflow, a range error occurs and `math.POS_HUGE_VAL` or `math.NEG_HUGE_VAL` (with the same sign as *x*) will be returned.

Errors

If the *x* argument is subnormal or if the result would cause an overflow, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.sinh(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.sinh(-1);
```

math.sqrt()

Computes the square root of its argument *x*.

Parameters

x - Floating point value.

Return value

Upon successful completion, this function returns the square root of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0 or `math.POS_INFINITY`, *x* will be returned.

If *x* is a finite value < -0 or `math.NEG_INFINITY`, a domain error occurs and a NaN will be returned.

Errors

- If the *x* argument is < -0 or `math.NEG_INFINITY`, then `fastly.error` will be set to `EDOM`.

Format

```
FLOAT
math.sqrt(FLOAT x)
```

Examples

```
1  declare local var.fi FLOAT;
2  declare local var.fo FLOAT;
3
4  set var.fi = 9.0;
5  set var.fo = math.sqrt(var.fi);
```

math.tan()

Computes the tangent of its argument *x*, measured in radians.

Parameters

x - Floating point value representing an angle in radians.

Return value

Upon successful completion, this function returns the tangent of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, *x* will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, a domain error occurs and a NaN will be returned.

If *x* is subnormal, a range error occurs and *x* will be returned.

If the result would cause an overflow, a range error occurs and `math.tan()` will return `math.POS_HUGE_VAL` or `math.NEG_HUGE_VAL`, with the same sign as the result of the function.

Errors

- If the *x* argument is `math.POS_INFINITY` or `math.NEG_INFINITY`, then `fastly.error` will be set to `EDOM`.
- If the *x* argument is subnormal or if the result overflows, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.tan(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.tan(math.PI_4);
```

math.tanh()

Computes the hyperbolic tangent of its argument *x*.

Parameters

x - Floating point value representing a hyperbolic angle.

Return value

Upon successful completion, this function returns the hyperbolic tangent of *x*.

If *x* is `math.NAN`, a NaN will be returned.

If *x* is ±0, *x* will be returned.

If *x* is `math.POS_INFINITY` or `math.NEG_INFINITY`, ±1 will be returned.

If *x* is subnormal, a range error occurs and *x* will be returned.

Errors

If the *x* argument is subnormal, then `fastly.error` will be set to `ERANGE`.

Format

```
FLOAT
math.tanh(FLOAT x)
```

Examples

```
1  declare local var.fo FLOAT;
2
3  set var.fo = math.tanh(-1);
```

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

addr.extract_bits()

Extracts `bit_count` bits (at most 32) starting with the bit number `start_bit` from the given IPv4 or IPv6 address and return them in the form of a non-negative integer.

Bit numbering starts at 0 from the right-most end of the address (the lowest order bit in the last byte of the address is bit number 0). As this function extracts bits from the address, it copies them to form the integer. In the address from which it extracts bits, the lowest order bit extracted from the first byte (the right-most byte) will be copied to the lowest order bit in the resulting integer.

If this function goes past the highest order bit in the left-most byte in the address before completing the copying of `bit_count` bits, then it will leave the remaining high-order bits in the integer at zero.

The bit count can be, at most, 32. The start bit must be lower than 128. The bit count plus start bit must be, at most, 128. If the VCL using this function violates any of these three constraints, then it will be rejected at compilation time.

The start bit and bit count must be constant values.

IPv6 addresses are 128 bits and IPv4 addresses are 32 bits. This function behaves as if an IPv4 address were padded with zeros on the left to 128 bits. If this function is applied to an address that is neither IPV4 nor IPv6, then it will return 0.

Format

```
INTEGER
addr.extract_bits(IP, start_bit INTEGER, bit_count INTEGER)
```

Examples

```
if (addr.extract_bits(server.ip, 0, 8) == 7) {
  # received on an IPv4 address that ends in ".7" or an IPv6 address that ends in "07"
}
```

addr.is_ipv4()

Returns true if the address family of the given address is IPv4.

Format

```
BOOL
addr.is_ipv4(IP ip)
```

Examples

```
1  if (addr.is_ipv4(client.ip)) {
2    # the client connected over IPv4 */
3  }
```

addr.is_ipv6()

Returns true if the address family of the given address is IPv6.

Format

```
BOOL
addr.is_ipv6(IP ip)
```

Examples

```
1  if (addr.is_ipv6(client.ip)) {
2    # the client connected over IPv6 */
3  }
```

http_status_matches()

Determines whether the HTTP status matches or does not match any of the statuses in the supplied *fmt* string.

Returns true when the status matches any of the strings and returns false otherwise. If *fmt* is prefixed with `!`, returns true when the status does not match any of the strings and returns false if it does. Statuses in the string are separated by commas.

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

Format

```
BOOL
http_status_matches(INTEGER status, STRING fmt)
```

Examples

```
1  if (http_status_matches(beresp.status, "!200,301,302")) {
2    set obj.cacheable = 0;
3  }
```

if()

Implements a ternary operator for strings; if the expression is true, it returns `value-when-true`; if the expression is false, it returns `value-when-false`. When the `if(x, value-when-true, value-when-false);` argument is true, the `value-when-true` is returned. Otherwise, the `value-when-false` is returned.

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

Format

```
STRING
if(BOOL expression, STRING value-when-true, STRING value-when-false)
```

Examples

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

setcookie.get_value_by_name()

Returns a value associated with the `cookie_name` in the `Set-Cookie` header contained in the HTTP response indicated by `where`. An unset value is returned if cookie is not found or on error. In the `vcl_fetch` method, the `beresp` response is available. In `vcl_deliver` and `vcl_log`, the `resp` response is available.

If multiple cookies of the same name are present in the response, the value of the last one will be returned.

When this function does not have enough memory to succeed, the request is failed.

This function conforms to RFC6265.

Format

```
STRING
setcookie.get_value_by_name(ID where, STRING cookie_name)
```

Examples

```
set resp.http.MyValue = setcookie.get_value_by_name(resp, "myvalue");
```

std.collect()

Combines multiple instances of the same header into one. The headers are joined using the optional separator character parameter. If omitted, `,` is used. A space is automatically added after each separator.

Multiple Set-Cookie headers should not be combined into a single header as this might lead to unexpected results on the browser side.

Format

```
VOID
std.collect(STRING header [, STRING separator_character])
```

Examples

```
# For a request with these Cookie headers:
# Cookie: name1=value1
# Cookie: name2=value2
std.collect(req.http.Cookie, ";");
# req.http.Cookie is now "name1=value1; name2=value2"
```

subfield()

Provides a means to access subfields from a header like `Cache-Control`, `Cookie`, and `Edge-Control` or individual parameters from the query string.

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", ";")) {
  #....
}
```

```
set req.http.value-of-foo = subfield(req.url.qs, "foo", "&");
```

Miscellaneous Variables

bereq.url.basename

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

Type

`STRING`

Accessibility

Readable From

All subroutines

bereq.url.dirname

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

Type

`STRING`

Accessibility

Readable From

All subroutines

bereq.url.qs

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

Type

`STRING`

Accessibility

Readable From

All subroutines

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`.

Type

`STRING`

Accessibility

Readable From

All subroutines

beresp.backend.ip

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

Type

`IP`

Accessibility

Readable From

- `vcl_fetch`

beresp.backend.name

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

Type

`STRING`

Accessibility

Readable From

- `vcl_fetch`

beresp.backend.port

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

Type

`INTEGER`

Accessibility

Readable From

- `vcl_fetch`

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`.

Type

`RTIME`

Accessibility

Readable From

- `vcl_fetch`

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 for instructions on enabling this feature for your account.

Type

`BOOL`

Accessibility

Readable From

- `vcl_fetch`

beresp.pci

Specifies that content be cached in a manner that satisfies PCI DSS requirements. See our PCI compliance description for instructions on enabling this feature for your account.

Type

`BOOL`

Accessibility

Readable From

- `vcl_fetch`

client.ip

The IP address of the client making the request.

Type

`IP`

Accessibility

Readable From

All subroutines

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.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

client.requests

Tracks the number of requests received by Varnish over a persistent connection. Over an HTTP/2 connection, tracks the number of multiplexed requests.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

client.socket.pace

Ceiling rate in kilobytes per second for bytes sent to the client.

This rate accounts for header sizes and retransmits, so the application level rate might be different from the one set here.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

fastly.error

Contains the error code raised by the last function, otherwise *not set*.

States

- `EPARSENUM`: Number parsing failed.
- `ERANGE`: Numerical result out of range.
- `EREGRECUR`: Call to regex routine failed because of recursion limits.
- `EREGSUB`: Call to regex routine failed (generic).
- `ESESOOM`: Out of workspace memory.
- `EDOM`: Domain error. This occurs for a mathematical function that is not defined for a particular value. Formally, that value is not considered part of its input domain. For example, division by zero, or `var.x %= 5;` where `var.x` is a floating point infinity.
- `ESYNTHOOM`: Synthetic response overflow.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.backend.healthy

Whether or not this backend, or recursively any of the backends under this director, is considered healthy. The random director has the additional constraint that the `quorum` threshold must be met by the healthy backends under the director. The health state is determined by: healthcheck results, whether there is room for a new connection to be made to the backend based on the number of currently used connections and the backend's `max_connections` setting, and any applicable saintmode settings.

Type

BOOL

Accessibility

Readable From

- `vcl_deliver`
- `vcl_error`
- `vcl_fetch`
- `vcl_hash`
- `vcl_hit`
- `vcl_miss`
- `vcl_pass`
- `vcl_recv`

req.backend.is_cluster

True if this backend, or recursively any of the backends under this director, is a cluster backend. False otherwise.

Type

BOOL

Accessibility

Readable From

All subroutines

req.backend.is_origin

True if this backend, or recursively any of the backends under this director, is not a shield backend. False otherwise.

Type

BOOL

Accessibility

Readable From

- `vcl_fetch`
- `vcl_miss`
- `vcl_pass`

req.backend.is_shield

True if this backend, or recursively any of the backends under this director, is a shield backend. False otherwise.

Type

BOOL

Accessibility

Readable From

All subroutines

req.backend

The backend to use to service the request.

Type

`BACKEND`

Accessibility

Readable From

All subroutines

req.body.base64

Same as `req.body`, except the request body is encoded in Base64, which handles null characters and allows representation of binary bodies.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.body

The request body. Using this variable for binary data will truncate at the first null character. Limited to 8KB in size. Exceeding the limit results in the `req.body` variable being blank. The variable `req.postbody` is an alias for `req.body`.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.grace

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

Type

`RTIME`

Accessibility

Readable From

All subroutines

req.http.host

The full host name, without the path or query parameters.

Examples

For example, in the request `www.example.com/index.html?a=1&b=2`, `req.http.host` will contain `www.example.com`.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.is_ipv6

Indicates whether the request was made using IPv6 or not.

Type

`BOOL`

Accessibility

Readable From

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

req.restarts

Counts the number of times the VCL has been restarted.

Type

`INTEGER`

Accessibility

Readable From

All subroutines

req.url.basename

The file name specified in a URL.

Examples

In the request `www.example.com/1/hello.gif?foo=bar`, `req.url.basename` will contain `hello.gif`.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.url.dirname

The directories specified in a URL.

Examples

- In the request `www.example.com/1/hello.gif?foo=bar`, `req.url.dirname` will contain `/1`.
- In the request `www.example.com/5/inner/hello.gif?foo=bar`, `req.url.dirname` will contain `/5/inner`.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.url.ext

The file extension specified in a URL.

Examples

In the request `www.example.com/index.html?a=1&b=2`, `req.url.ext` will contain `html`.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.url.path

The full path, without any query parameters.

Examples

In the request `www.example.com/inner/index.html?a=1&b=2`, `req.url.path` will contain `/inner/index.html`.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.url.qs

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

Examples

In the request `www.example.com/index.html?a=1&b=2`, `req.url.qs` will contain `a=1&b=2`.

Type

`STRING`

Accessibility

Readable From

All subroutines

req.url

The full path, including query parameters.

Examples

In the request `www.example.com/index.html?a=1&b=2`, `req.url` will contain `/index.html?a=1&b=2`.

Type

`STRING`

Accessibility

Readable From

All subroutines

stale.exists

Specifies if a given object has stale content in cache. Returns `true` or `false`.

Type

`STRING`

Accessibility

Readable From

All subroutines

Query string manipulation

Examples

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

```
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()

Alias of `querystring.sort`.

Format

```
STRING
boltsort.sort(STRING url)
```

Examples

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

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

```
STRING
querystring.add(STRING, STRING, STRING)
```

Examples

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

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

```
STRING
querystring.clean(STRING)
```

Examples

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

querystring.filter_except()

Returns the given URL but only keeps the listed parameters.

Format

```
STRING
querystring.filter_except(STRING, STRING_LIST)
```

Examples

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

querystring.filter()

Returns the given URL without the listed parameters.

Format

```
STRING
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()

Returns the separator needed by the `querystring.filter()` and `querystring.filter_except()` functions.

Format

```
STRING
querystring.filtersep()
```

Examples

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

querystring.globfilter_except()

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

Format

```
STRING
querystring.globfilter_except(STRING, STRING)
```

Examples

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

querystring.globfilter()

Returns the given URL without the parameters matching a glob.

Format

```
STRING
querystring.globfilter(STRING, STRING)
```

Examples

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

querystring.regfilter_except()

Returns the given URL but only keeps the parameters matching a regular expression. Groups within the regular expression are treated as if they were written as non-capturing groups. For example:

```
1  if (req.url.qs ~ "key-(?:[0-9]|\w)=(.*)-(.*)") { # captures to re.group.1 and re.group.2
2    set req.url = querystring.regfilter_except(req.url, "key-([0-9]|\w)"); # does not capture
3    set req.http.X-Key-1 = re.group.1;
4    set req.http.X-Key-2 = re.group.2;
5  }
```

The `"key-([0-9]|\w)"` pattern shown here behaves as if it were written as a non-capturing group, `"key-(?:[0-9]|\w)"`, ensuring the contents of `re.group.1` and `re.group.2` are not affected by the call to `querystring.regfilter_except()`.

Format

```
STRING
querystring.regfilter_except(STRING, STRING)
```

Examples

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

querystring.regfilter()

Returns the given URL without the parameters matching a regular expression. Groups within the regular expression are treated as if they were written as non-capturing groups. For example:

```
1  if (req.url.qs ~ "key-(?:[0-9]|\w)=(.*)-(.*)") { # captures to re.group.1 and re.group.2
2    set req.url = querystring.regfilter(req.url, "key-([0-9]|\w)"); # does not capture
3    set req.http.X-Key-1 = re.group.1;
4    set req.http.X-Key-2 = re.group.2;
5  }
```

The `"key-([0-9]|\w)"` pattern shown here behaves as if it were written as a non-capturing group, `"key-(?:[0-9]|\w)"`, ensuring the contents of `re.group.1` and `re.group.2` are not affected by the call to `querystring.regfilter()`.

Format

```
STRING
querystring.regfilter(STRING, STRING)
```

Examples

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

querystring.remove()

Returns the given URL with its query-string removed.

Format

```
STRING
querystring.remove(STRING)
```

Examples

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

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

```
STRING
querystring.set(STRING, STRING, STRING)
```

Examples

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

querystring.sort()

Returns the given URL with its query-string sorted. For example, `querystring.sort("/foo?b=1&a=2&c=3");` returns `"/foo?a=2&b=1&c=3"`.

Format

```
STRING
querystring.sort(STRING)
```

Examples

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

Randomness

WARNING: We use BSD random number functions from the GNU C Library, not true randomizing sources. These VCL functions should not be used for 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_-`.

```
1  sub vcl_deliver {
2    set resp.http.Foo = "randomstuff=" randomstr(10);
3    set resp.http.Bar = "morsecode=" randomstr(50, ".-"); # 50 dots and dashes
4  }
```

Random content cookies in pure VCL

```
1  sub vcl_deliver {
2    add resp.http.Set-Cookie = "somerandomstuff=" randomstr(10) "; expires=" now + 180d "; path=/;";
3  }
```

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.

```
1  sub vcl_recv {
2    if (randombool(1, 4)) {
3      set req.http.X-AB = "A";
4    } else {
5      set req.http.X-AB = "B";
6    }
7  }
```

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:

```
1  if (randombool(std.atoi(req.http.Some-Header), 100))  {
2    # do something
3  }
```

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`:

```
1  if (randombool_seeded(50, 100, 12345)) {
2    set resp.http.Seeded-Value = "yes";
3  } else {
4    set resp.http.Seeded-Value = "no";
5  }
```

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

randombool_seeded()

Identical to randombool, except takes an additional parameter, which is used to seed the random number generator.

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

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

Format

```
BOOL
randombool_seeded(INTEGER numerator, INTEGER denominator, INTEGER seed)
```

Examples

```
1  set req.http.my-hmac = digest.hmac_sha256("sekrit", req.http.X-Token);
2  set req.http.hmac-chopped = regsub(req.http.my-hmac, "^(..........).*$","\1");
3  if (randombool_seeded(5,100,std.strtol(req.http.hmac-chopped ,16))) {
4    set req.http.X-Allowed = "true";
5  } else {
6    set req.http.X-Allowed = "false";
7  }
```

randombool()

Returns a random, boolean value. The result is true when, given a pseudorandom number `r`, `(RAND_MAX * numerator) > (r * denominator)`.

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

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

Format

```
BOOL
randombool(INTEGER numerator, INTEGER denominator)
```

Examples

```
1  if (randombool(1, 10)) {
2    set req.http.X-ABTest = "A";
3  } else {
4    set req.http.X-ABTest = "B";
5  }
```

randomint_seeded()

Identical to 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.

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

Format

```
INTEGER
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()

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.

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

Format

```
INTEGER
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";
}
```

randomstr()

Returns a random string of length `len` containing characters from the supplied string `characters`.

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

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

Format

```
STRING
randomstr(INTEGER len, STRING characters)
```

Examples

```
set req.http.X-RandomHexNum = randomstr(8, "1234567890abcdef");
```

Segmented Caching

Segmented Caching Variables

fastly.segmented_caching.autopurged

Whether an inconsistency encountered during Segmented Caching processing led to the system automatically enqueuing a purge request.

Type

`BOOL`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.block_number

A zero-based counter identifying the file fragment being processed. This variable will evaluate to `-1` in cases when it is not applicable, such as when Segmented Caching is not enabled for the request.

Type

`INTEGER`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.cancelled

Whether Segmented Caching processing was enabled and cancelled by a non-206 response.

Type

`BOOL`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.client_req.is_open_ended

Whether the client's request leaves the upper bound of the range open. This variable will evaluate to `false` when Segmented Caching is not enabled.

Type

`BOOL`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.client_req.is_range

Whether the client's request is a range request. This variable with evaluate to `false` when Segmented Caching is not enabled for the request (even if `req.http.Range` is present).

Type

`BOOL`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.client_req.range_high

The upper bound of the client's requested range. This variable will evaluate to `-1` in cases when it is not applicable, such as when Segmented Caching is not enabled for the request. It will evaluate to `9223372036854775807` (`2^63-1`) for an open-ended requested range (when `fastly.segmented_caching.client_req.is_open_ended` is true).

Type

`INTEGER`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.client_req.range_low

The lower bound of the client's requested range. This variable will evaluate to `-1` in cases when it is not applicable, such as when Segmented Caching is not enabled for the request.

Type

`INTEGER`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.completed

Whether Segmented Caching processing was enabled and cancelled by a non-206 response.

Type

`BOOL`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.error

The reason why Segmented Caching processing failed. This variable will evaluate to `NULL` if Segmented Caching was not enabled, or if Segmented Caching processing completed successfully or was cancelled by a non-206 response.

Type

`STRING`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.failed

Whether Segmented Caching processing was enabled and ended in a failure. When this variable evaluates to `true`, the variable `fastly.segmented_caching.error` will evaluate to a string describing the nature of the failure.

Type

`BOOL`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.is_inner_req

Whether VCL is running in the context of a sub-request that is retrieving a fragment of a file. If using the default 1MB object size, there will be a log line on every 1MB request back to origin.

Type

`BOOL`

Accessibility

Readable From

- `vcl_log`

fastly.segmented_caching.is_outer_req

Whether VCL is running in the context of a request that is assembling file fragments into a response.

Type

BOOL

Accessibility

Readable From

- vcl_log

fastly.segmented_caching.obj.complete_length

The size of the whole file in bytes. The information comes from the Content-Range response header field in the first fragment accessed while assembling the response. This variable will evaluate to -1 in cases when it is not applicable, such as when Segmented Caching is not enabled for the request.

Type

INTEGER

Accessibility

Readable From

- vcl_log

fastly.segmented_caching.rounded_req.range_high

The upper bound of the rounded block being processed. This variable will evaluate to -1 in cases when it is not applicable, such as when Segmented Caching is not enabled for the request.

Type

INTEGER

Accessibility

Readable From

- vcl_log

fastly.segmented_caching.rounded_req.range_low

The lower bound of the rounded block being processed. This variable will evaluate to -1 in cases when it is not applicable, such as when Segmented Caching is not enabled for the request.

Type

INTEGER

Accessibility

Readable From

- vcl_log

fastly.segmented_caching.total_blocks

The number of fragments needed for assembling this response. This variable will evaluate to -1 in cases when it is not applicable, such as when Segmented Caching is not enabled for the request.

Type

INTEGER

Accessibility

Readable From

- vcl_log

Server

Server Variables

server.datacenter

A code representing one of Fastly's POP locations.

Type

`STRING`

Accessibility

Readable From

All subroutines

server.hostname

Hostname of the server (e.g., `cache-jfk1034`).

Type

`STRING`

Accessibility

Readable From

All subroutines

server.identity

Same as `server.hostname` but also explicitly includes the datacenter name (e.g., `cache-jfk1034-JFK`).

Type

`STRING`

Accessibility

Readable From

All subroutines

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-Central`	the central European continent
`EU-East`	the eastern European continent
`EU-West`	the western European continent
`North-America`	Canada
`SA-East`	eastern South America
`SA-North`	northern 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

Type

`STRING`

Accessibility

Readable From

All subroutines

Size

Size Variables

bereq.body_bytes_written

Total body bytes written to a backend. Does not include header bytes.

Type

`INTEGER`

Accessibility

Readable From

- `vcl_fetch`
- `vcl_deliver`
- `vcl_log`

bereq.header_bytes_written

Total header bytes written to a backend.

Type

`INTEGER`

Accessibility

Readable From

- `vcl_fetch`
- `vcl_deliver`
- `vcl_log`

req.body_bytes_read

Total body bytes read from the client generating the request.

Type

`STRING`

Accessibility

Readable From

- `vcl_deliver`

- `vcl_log`

req.bytes_read

Total bytes read from the client generating the request.

Type

`STRING`

Accessibility

Readable From

- `vcl_deliver`
- `vcl_log`

req.header_bytes_read

Total header bytes read from the client generating the request.

Type

`STRING`

Accessibility

Readable From

All subroutines

resp.body_bytes_written

Body bytes to send to the client in the response.

Type

`STRING`

Accessibility

Readable From

- `vcl_log`

resp.bytes_written

Total bytes to send to the client in the response.

Type

`STRING`

Accessibility

Readable From

- `vcl_log`

resp.completed

Whether the response completed successfully or not.

Type

`BOOL`

Accessibility

Readable From

- `vcl_log`

resp.header_bytes_written

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

Type

`STRING`

Accessibility

Readable From

- `vcl_log`

String manipulation

String manipulation Functions

cstr_escape()

Escapes bytes from a string using C-style escape sequences.

The escaping rules in priority order are as follows:

1. if the byte is the doublequote (0x22), it is escaped as `\"` (backslash doublequote)
2. if the byte is the backslash (0x5C), it is escaped as `\\` (double backslash)
3. if the byte is one of the following control characters, it is escaped as follows:
   - `\b` (0x08, backspace)
   - `\t` (0x09, horizontal tab)
   - `\n` (0x0A, newline)
   - `\v` (0x0B, vertical tab)
   - `\r` (0x0D, carriage return)
4. if the byte is less than or equal to 0x1F, or it is greater or equal to 0x7F (in other words, a control character not explicitly listed above), it is escaped as `\xHH` where HH is the hexadecimal value of the byte
5. if none of the above matched, the byte is passed through as-is: for example `a` for 0x61

> **TIP:** If you are escaping JSON strings, use `json.escape()` instead.

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

Format

```
STRING
cstr_escape(STRING string)
```

Examples

```
# var.escaped is set to: city="london"
declare local var.escaped STRING;
set var.escaped = "city=%22" + cstr_escape(client.geo.city.ascii) + "%22";
```

json.escape()

Escapes characters of a UTF-8 encoded Unicode string using JSON-style escape sequences.

Format

```
STRING
json.escape(STRING string)
```

Examples

```
declare local var.json STRING;
set var.json = "{%22city%22: %22" + json.escape(client.geo.city.utf8) + "%22}";
# var.json is now e.g. {"city": "london"}
```

regsub()

Replaces the first occurrence of `pattern`, which is a Perl-compatible regular expression, in `input` with `replacement`. If no match is found, no replacement is made. Calls to `regsub` do not set `re.group.*`.

This function may fail to make a replacement if the regular expression recurses too heavily. Such a situation may occur with lookahead and lookbehind assertions, or other recursing non-regular expressions. In this case, `fastly.error` is set to `EREGRECUR`.

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

Format

```
STRING
regsub(STRING input, STRING pattern, STRING replacement)
```

Examples

```
# The following example deletes any query string parameters
set req.url = regsub(req.url, "\?.*$", "");
```

regsuball()

Replaces all occurrences of `pattern`, which may be a Perl-compatible regular expression, in `input` with `replacement`. If no matches are found, no replacements are made.

Once a replacement is made, substitutions continue from the end of the replaced buffer. Therefore, `regsuball("aaa", "a", "aa")` will return a string "aaaaaa" instead of recursing indefinitely.

This function may fail to make a replacement if the regular expression recurses too heavily. Such a situation may occur with lookahead and lookbehind assertions, or other recursing non-regular expressions. In this case, `fastly.error` is set to `EREGRECUR`.

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

Format

```
STRING
regsuball(STRING input, STRING pattern, STRING replacement)
```

Examples

```
set req.url = regsuball(req.url, "\+", "%2520");
```

std.anystr2ip()

Converts the string `addr` to an IP address (IPv4 or IPv6). If conversion fails, `fallback` will be returned.

This function accepts a wider range of formats than `std.str2ip()`: Each number may be specified in hexadecimal (`0x...`), octal (`0...`), or decimal format, and there may be fewer than four numbers, in which case the last number is responsible for the remaining bytes of the IP. For example, `0x8.010.2056` is equivalent to `8.8.8.8`.

We recommend using a fallback IP address that's meaningful for your particular Fastly service.

Format

```
IP
std.anystr2ip(STRING addr, STRING fallback)
```

Examples

```
if (std.anystr2ip("0xc0.0.01001", "192.0.2.2") ~ my_acl) {
  ...
}
```

std.atof()

Takes a string (which represents a float) as an argument and returns its value. Behaves as if calling `std.strtof()` with a base of 10.

Format

```
FLOAT
std.atof(STRING s)
```

Examples

```
if (std.atof(req.http.X-String) > 21.82) {
  set req.http.X-TheAnswer = "Found";
}
```

std.atoi()

Takes a string (which represents an integer) as an argument and returns its value. Behaves as if calling `std.strtol()` with a base of 10.

Format

```
INTEGER
std.atoi(STRING s)
```

Examples

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

std.ip()

An alias of `std.str2ip()`.

We recommend using a fallback IP address that's meaningful for your particular Fastly service.

Format

```
IP
std.ip(STRING addr, STRING fallback)
```

Examples

```
if (std.ip(req.http.Fastly-Client-IP, "192.0.2.2") ~ my_acl) {
  ...
}
```

std.ip2str()

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

Format

```
STRING
std.ip2str(IP ip)
```

Examples

```
if (std.ip2str(std.str2ip(req.http.Fastly-Client-IP, "192.0.2.2")) ~ my_acl) {
  ...
}
```

std.prefixof()

True if the string `s` begins with the string `begins_with`. An empty string is not considered a prefix.

Returns false otherwise.

Format

```
BOOL
std.prefixof(STRING s, STRING begins_with)
```

Examples

```
set req.http.X-ps = std.prefixof("greenhouse", "green");
```

std.str2ip()

Converts the string representation of an IP address (IPv4 or IPv6) into an `IP type`. If conversion fails, the fallback will be returned. The string must be a numeric IP address representation in the standard format such as `192.0.2.2` and `2001:db8::1`. This function does not support looking up an IP address by name.

We recommend using a fallback IP address that's meaningful for your particular Fastly service.

Format

```
IP
std.str2ip(STRING addr, STRING fallback)
```

Examples

```
1  if (std.str2ip(req.http.Fastly-Client-IP, "192.0.2.2") ~ my_acl) {
2    ...
3  }
```

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
std.strlen(STRING s)
```

Examples

```
1  if (std.strlen(req.http.Cookie) > 1024) {
2    unset req.http.Cookie;
3  }
```

std.strpad()

This function constructs a string containing the input string `s` padded out with `pad` to produce a string of the given `width`. The padding string `pad` is repeated as necessary and cut short when `width` is reached.

Note that `width` is given in bytes and this function will cut short paddings with multi-byte encodings.

Negative `width` left-justifies `s` by placing padding to the right. Positive `width` right-justifies `s` by placing padding to the left. If `width` is less than or equal to the length of `s`, then no padding is performed.

If `pad` is the empty string, then no padding is performed and the unmodified string `s` is returned.

Format

```
STRING
std.strpad(STRING s, INTEGER width, STRING pad)
```

Examples

```
set var.s = std.strpad("abc", -10, "-="); # produces "abc-=-=-=-"
```

```
set var.s = std.strpad("abc", 10, "-="); # produces "-=-=-=-abc"
```

```
1  declare local var.n INTEGER;
2  set var.n = std.strlen("abcd");
3  set var.n *= 3;
4  set var.s = std.strpad("", var.n, "abcd"); # repeat "abcd" three times
```

std.strrep()

Repeats the given string `n` times. If `n` is a negative value, it is taken to mean zero.

Format

```
STRING
std.strrep(STRING s, INTEGER n)
```

Examples

```
set var.s = std.strrep("abc", 3); # produces "abcabcabc"
```

std.strrev()

Reverses the given string. This function does not support UTF-8 encoded strings.

Errors

This function will set `fastly.error` to `EUTF8` if the input string *s* is UTF-8 encoded.

Format

```
STRING
std.strrev(STRING s)
```

Examples

```
set var.s = std.strrev("abc"); # produces "cba"
```

std.strstr()

Returns the part of `haystack` string starting from and including the first occurrence of `needle` until the end of `haystack`.

Format

```
STRING
std.strstr(STRING haystack, STRING needle)
```

Examples

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

std.strtof()

Converts the string *s* to a float value with the given base *base*. The value *base* must be a constant integer expression (variables are not allowed).

The following string formats are supported for finite values:

- Decimal (base 10) floating point syntax. For example, `1.2`, `-1.2e-3`.
- Hexadecimal (base 16) floating point syntax. For example, `0xA.B`, `0xA.Bp-3`.

The syntax for these values corresponds to the syntax for VCL `FLOAT` literals in base 10 and 16 respectively. See VCL Types for details of the `FLOAT` literal syntax.

Supported bases are 0, 10, or 16.

A base of 0 causes the base to be automatically determined from the string format. In this case, a `0x` prefix indicates hex (base 16), and otherwise the base is taken as decimal (base 10).

The syntax is required to match with a corresponding prefix when an explicit base is given. That is, for base 16, the `0x` prefix must be present. Likewise for base 10, the `0x` prefix must be absent.

Numbers are parsed with a rounding mode of *round to nearest with ties away from zero*.

In addition to finite values, the following special string formats are supported:

- `NaN`: NaN may be produced by the special format `NaN`. Note that only one NaN representation is produced.
- `inf`, `+inf`, `-inf`: Positive and negative infinities may be produced by the special format `inf` with an optional preceding +/- sign.

The NaN and infinity special formats are case sensitive.

No whitespace is permitted by `std.strtof`.

On error, `fastly.error` is set.

Format

```
INTEGER
std.strtof(STRING s, INTEGER base)
```

Examples

```
declare local var.pi FLOAT;
set var.pi = std.strtof(req.http.PI, 10);
if (var.pi >= 3.14 && var.pi <= 3.1416) {
  set req.http.X-PI = "Close enough";
}
```

std.strtol()

Converts the string *s* to an integer value. The value *base* must be a constant integer expression or integer-returning function.

The following string formats are supported:

- Decimal (base 10) integer syntax. For example, `123`, `-4`.
- Hexadecimal (base 16) integer syntax. For example, `0xABC`, `-0x0`.
- Octal (base 8) integer syntax. For example, `0`, `0123`.

The syntax for integers extends the syntax for VCL `INTEGER` literals in base 10 and 16 respectively. See VCL Types for details of the `INTEGER` literal syntax for these bases.

Supported bases are 2 - 36, inclusive and the special value 0. For bases over 10, the alphabetic digits are case insensitive.

A base of 0 causes the base to be automatically determined from the string format. In this case, a `0x` prefix indicates hex (base 16), a prefix of `0` indicates octal (base 8), and otherwise the base is taken as decimal (base 10).

When an explicit base is specified, the hexadecimal prefix of `0x` and the octal prefix of `0` are not required.

Whitespace and trailing characters are permitted and have no effect on the value produced.

If the base is outside the range, or the number exceeds the range of a signed integer, `fastly.error` is set to `ERANGE`. If the number could not be parsed, `fastly.error` is set to `EPARSENUM`.

On error, `fastly.error` is set.

Format

```
INTEGER
std.strtol(STRING s, INTEGER base)
```

Examples

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

std.suffixof()

True if the string `s` ends with the string `ends_with`. An empty string is not considered a suffix.

Returns false otherwise.

Format

```
BOOL
std.suffixof(STRING s, STRING ends_with)
```

Examples

```
set req.http.X-ss = std.suffixof("rectangles", "angles");
```

std.tolower()

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

Format

```
STRING
std.tolower(STRING_LIST s)
```

Examples

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

std.toupper()

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

Format

```
STRING
std.toupper(STRING_LIST s)
```

Examples

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

substr()

Returns a substring of the byte string *s*, starting from the byte *offset*, of byte *length*. The substring is a copy of the original bytes.

The *length* parameter is optional. If it's not specified, it means until the end of the string.

The *offset* parameter is zero-based. For example, `substr("abcdefg", 0, 3)` is `"abc"`.

If the requested range is partially outside the string *s*, the returned string is truncated. For example, `substr("abcdefg", 5, 3)` is `"fg"`.

If the requested range is completely outside the string *s*, an *unset* value is returned. For example, `substr("abc", 4, 2)` returns an *unset* value, the edge case `substr("abc", 3, 2)` being `""`.

A negative *offset* counts backwards from the end of the string *s*. For example, `substr("abcdefg", -3, 2)` is `"ef"`.

A negative *length* counts backwards from the end of the string *s* with the *offset* taken into account. For example, `substr("abcdefg", 1, -3)` is `"bcd"` and `substr("abcdefg", -4, -3)` is `"de"`.

An *unset* value is also returned in the extreme edge cases of the *offset* or *length* causing integer overflows.

NOTE: substr() does not correctly handle UTF-8 encoded Unicode strings because byte offsets and lengths are likely to result in invalid UTF-8. Use `utf8.substr()` to handle UTF-8 encoded Unicode strings.

Format

```
STRING
substr(STRING s, INTEGER offset [, INTEGER length])
```

Examples

```
log "left=" substr("foobar", 0, 3)
log "middle=" substr("foobar", 2, 3)
log "right=" substr("foobar", -3)
```

urldecode()

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

Format

```
STRING
urldecode(STRING input)
```

Examples

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

urlencode()

Encodes a string for use in a URL. This is also known as percent-encoding. For example, `urlencode("hello world");` will return `"hello%20world"`.

Format

```
STRING
urlencode(STRING input)
```

Examples

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

utf8.codepoint_count()

Returns the number of UTF-8 encoded Unicode code points in the string *s*. Returns zero if the string does not contain valid UTF-8.

Format

```
STRING
utf8.codepoint_count(STRING s)
```

utf8.is_valid()

Returns true if the string *s* contains valid UTF-8 and returns false if it does not contain valid UTF-8. An empty string is considered valid.

Format

```
BOOL
utf8.is_valid(STRING s)
```

utf8.strpad()

Like `std.strpad()` except `count` gives the number of unicode code points for the output string rather than bytes.

Errors

This function requires the input strings `s` and `pad` to be UTF-8 encoded. If they are not, `fastly.error` will be set to `EUTF8`.

Format

```
STRING
utf8.strpad(STRING s, INTEGER count, STRING pad)
```

Examples

```
1  utf8.strpad("abc", 7, "🌸🌼"); # gives "🌸🌼🌸🌼abc", seven code points total
2  std.strpad("abc", 7, "🌸🌼"); # gives "🌸abc" because 🌸 is four bytes
```

utf8.substr()

Returns a substring of the UTF-8 string *s*, starting from the Unicode code point *offset*, of Unicode code point *length*. The substring is a copy of the original bytes.

For example, `substr("%u{3b1}%u{3b2}%u{3b3}", 1, 1)` is `"β"`. See `substr()` for the exact semantics of the *offset* and *length*.

If the input string is not valid UTF-8, an *unset* value is returned.

> **NOTE:** UTF-8 allows you to combine characters, which are separate code points. While `utf8.substr()` correctly honors the Unicode code point boundaries, however, requesting a substring of several of them may not necessarily represent a meaningful grapheme cluster.

Format

```
STRING
utf8.substr(STRING s, INTEGER offset [, INTEGER length])
```

Table

Tables are declared as follows:

```
1  table <ID> {
2    "key1":    "value 1",
3    {"key2"}:  {"value 2"},
4  }
```

Either short-form or long-form strings are supported, as illustrated in the above example. The trailing comma after the final value is optional, but supported.

Table Functions

table.lookup()

Look up the key `key` in the table `ID`. When the key is present, its associated value will be returned. When the key is absent, the value returned is *not_set*.

When a third STRING argument is provided, the lookup function behaves as it would normally, except when a key is absent, the *default* value is returned instead.

Format

```
STRING
table.lookup(ID, STRING key [, STRING default])
```

Examples

```
table redirects {
  "/foo": "/bar",
  "/bat": "/baz",
}
set req.http.X-Redirect = table.lookup(redirects, req.url);
if (req.http.X-Redirect) {
  error 302 "Found";
}
```

```
table geoip_lang {
  "US": "en-US",
  "FR": "fr-FR",
  "NL": "nl-NL",
}
if (!req.http.Accept-Language) {
  set req.http.Accept-Language = table.lookup(geoip_lang, geoip.country_code, "en-US");
}
```

TLS and HTTP/2

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 as HTTP/1.1 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.disable_header_compression()

Sets a flag to disable HTTP/2 header compression on one or many response headers to the client. Field names are case insensitive.

Calling this function will save space in the dynamic table for other, more reusable, headers. Likewise, calling this function will not put sensitive header field values at risk by compressing them.

By default, we disable compression for `Cookie` or `Set-Cookie` headers.

Format

```
VOID
h2.disable_header_compression(STRING header)
```

Examples

```
h2.disable_header_compression("Authorization");
h2.disable_header_compression("Authorization", "Secret");
```

h2.push()

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

Format

```
VOID
h2.push(STRING resource)
```

Examples

```
if (fastly_info.is_h2 && req.url == "/") {
  h2.push("/assets/jquery.js");
}
```

TLS and HTTP/2 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`

Accessibility

Readable From

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

fastly_info.h2.stream_id

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

Type

`INTEGER`

Accessibility

Readable From

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

fastly_info.is_h2

Whether or not the request was made using HTTP/2.

Type

`BOOL`

Accessibility

Readable From

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

tls.client.cipher

The cipher suite used to secure the client TLS connection. The value returned will be consistent with the OpenSSL Name.

Examples

`"ECDHE-RSA-AES128-GCM-SHA256"`

Type

`STRING`

Accessibility

Readable From

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

tls.client.ciphers_list_sha

A SHA-1 digest of the raw buffer containing the list of supported ciphers, represented in Base64.

Type

`STRING`

Accessibility

Readable From

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

tls.client.ciphers_list_txt

The list of ciphers supported by the client, rendered as text, in a colon-separated list.

Type

`STRING`

Accessibility

Readable From

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

tls.client.ciphers_list

The list of ciphers supported by the client, as sent over the network, hex encoded.

Type

`STRING`

Accessibility

Readable From

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

tls.client.ciphers_sha

A SHA-1 of the cipher suite identifiers sent from the client as part of the TLS handshake, represented in Base64.

Type

`STRING`

Accessibility

Readable From

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

tls.client.protocol

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

Type

`STRING`

Accessibility

Readable From

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

tls.client.servername

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

Type

`STRING`

Accessibility

Readable From

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

tls.client.tlsexts_list_sha

A SHA-1 digest of the TLS extensions supported by the client as little-endian, 16-bit integers, represented in Base64.

Type

`STRING`

Accessibility

Readable From

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

- `vcl_log`

tls.client.tlsexts_list_txt

The list of TLS extensions supported by the client, rendered as text in a colon-separated list. The value returned will be consistent with the IANA Cipher Suite Name.

Type

`STRING`

Accessibility

Readable From

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

tls.client.tlsexts_list

The list of TLS extensions supported by the client as little-endian, 16-bit, unsigned integers, hex encoded.

Type

`STRING`

Accessibility

Readable From

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

tls.client.tlsexts_sha

A SHA-1 of the TLS extension identifiers sent from the client as part of the TLS handshake, represented in Base64.

Type

`STRING`

Accessibility

Readable From

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

UUID

UUID Functions

uuid.dns()

Returns the RFC4122 identifier of DNS namespace, namely the constant `"6ba7b810-9dad-11d1-80b4-00c04fd430c8"`.

Format

```
STRING
uuid.dns()
```

Examples

```
declare local var.dns STRING;
set var.dns = uuid.version3(uuid.dns(), "www.example.com");
# var.dns is now "5df41881-3aed-3515-88a7-2f4a814cf09e"
```

uuid.is_valid()

Returns true if the string holds a textual representation of a valid UUID (per RFC4122). False otherwise.

Format

```
BOOL
uuid.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()

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

Format

```
BOOL
uuid.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()

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

Format

```
BOOL
uuid.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()

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

Format

```
BOOL
uuid.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()

Returns the RFC4122 identifier of ISO OID namespace, namely the constant `"6ba7b812-9dad-11d1-80b4-00c04fd430c8"`.

Format

```
STRING
uuid.oid()
```

Examples

```
declare local var.oid STRING;
set var.oid = uuid.version3(uuid.oid(), "2.999");
# var.oid is now "31cb1efa-18c4-3d19-89ba-df6a74ddbd1d"
```

uuid.url()

Returns the RFC4122 identifier of URL namespace, namely the constant `"6ba7b811-9dad-11d1-80b4-00c04fd430c8"`.

Format

```
STRING
uuid.url()
```

Examples

```
declare local var.url STRING;
set var.url = uuid.version3(uuid.url(), "https://www.example.com/");
# var.url is now "7fed185f-0864-319f-875b-a3d5458e30ac"
```

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
uuid.version3(STRING namespace, STRING name)
```

Examples

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

uuid.version4()

Returns a UUID based on random number generator output.

Format

```
STRING
uuid.version4()
```

Examples

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

uuid.version5()

Derives a UUID corresponding to `name` within the given `namespace` using SHA-1 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
uuid.version5(STRING namespace, STRING name)
```

Examples

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

uuid.x500()

Returns the RFC4122 identifier of X.500 namespace, namely the constant `"6ba7b812-9dad-11d1-80b4-00c04fd430c8"`.

Format

```
STRING
uuid.x500()
```

Examples

```
declare local var.x500 STRING;
set var.x500 = uuid.version3(uuid.x500(), "CN=Test User 1, O=Example Organization, ST=California, C=US");
# var.x500 is now "addf5e97-9287-3834-abfd-7edcbe7db56f"
```

Guides

§ Custom VCL

Creating custom VCL

Fastly Varnish syntax is specifically compatible with Varnish 2.1.5. 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.** 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 will not be honored** and the expected behavior may change.

IMPORTANT: Personal data should not be incorporated into VCL. Our Compliance and Law FAQ describes in detail how Fastly handles personal data privacy.

Inserting custom VCL in Fastly's VCL boilerplate

DANGER: Start with the VCL boilerplate and 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. You must 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.

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 instead.

```
sub vcl_recv {
#FASTLY recv

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

  return(lookup);
}

sub vcl_fetch {
#FASTLY fetch

  if ((beresp.status == 500 || beresp.status == 503) && req.restarts < 1 && (req.method == "GET" || req.method == "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-Control ~ "(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
}
```

```
4
7
4
8
4
9
5
0
5
1
5
2
5
3
5
4
5
5
5
6
5
7
5
8
5
9
6
0
6
1
6
2
6
3
6
4
6
5
6
6
6
7
6
8
6
9
7
0
7
1
7
2
7
3
7
4
7
5
7
6
7
7
7
8
```

Uploading custom VCL

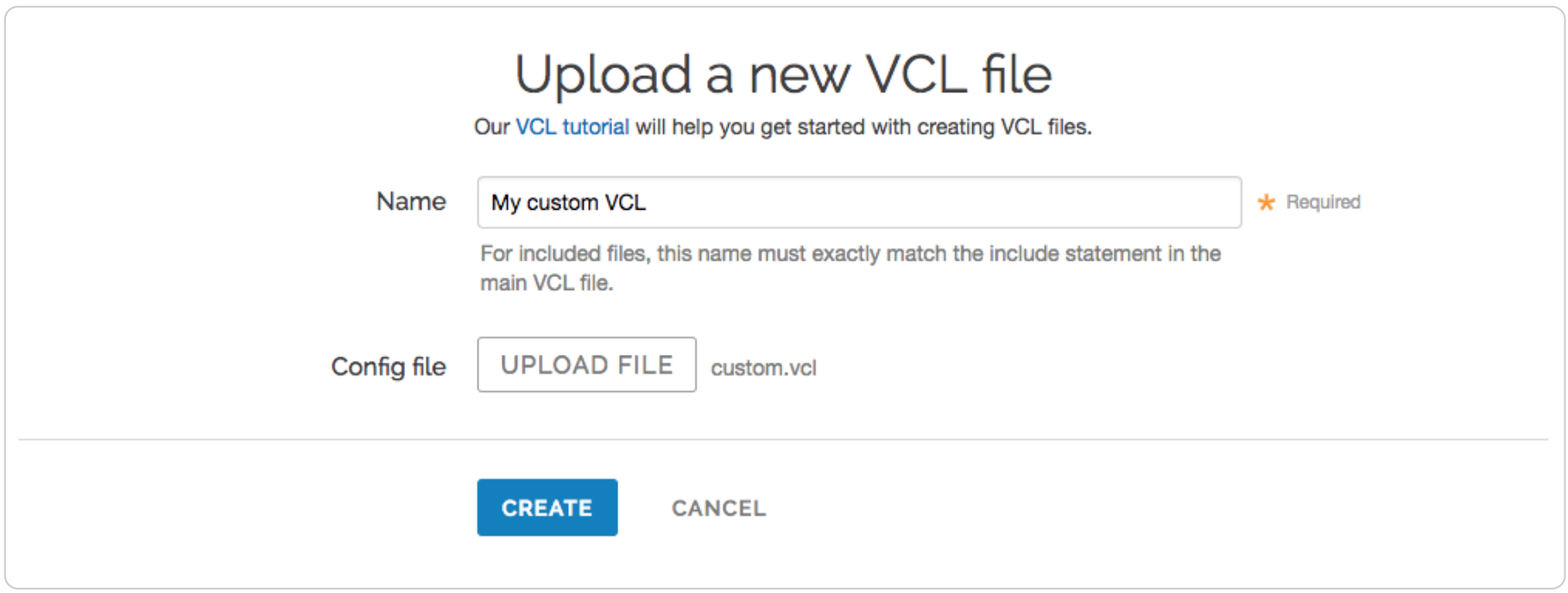
Fastly allows you create your own Varnish Configuration Language (VCL) files with specialized configurations. By uploading custom VCL files, you can use custom VCL and Fastly VCL together at the same time. Keep in mind that your custom VCL always takes precedence over VCL generated by Fastly.

IMPORTANT: Personal data should not be incorporated into VCL. Our Compliance and Law FAQ 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 **Edit 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.

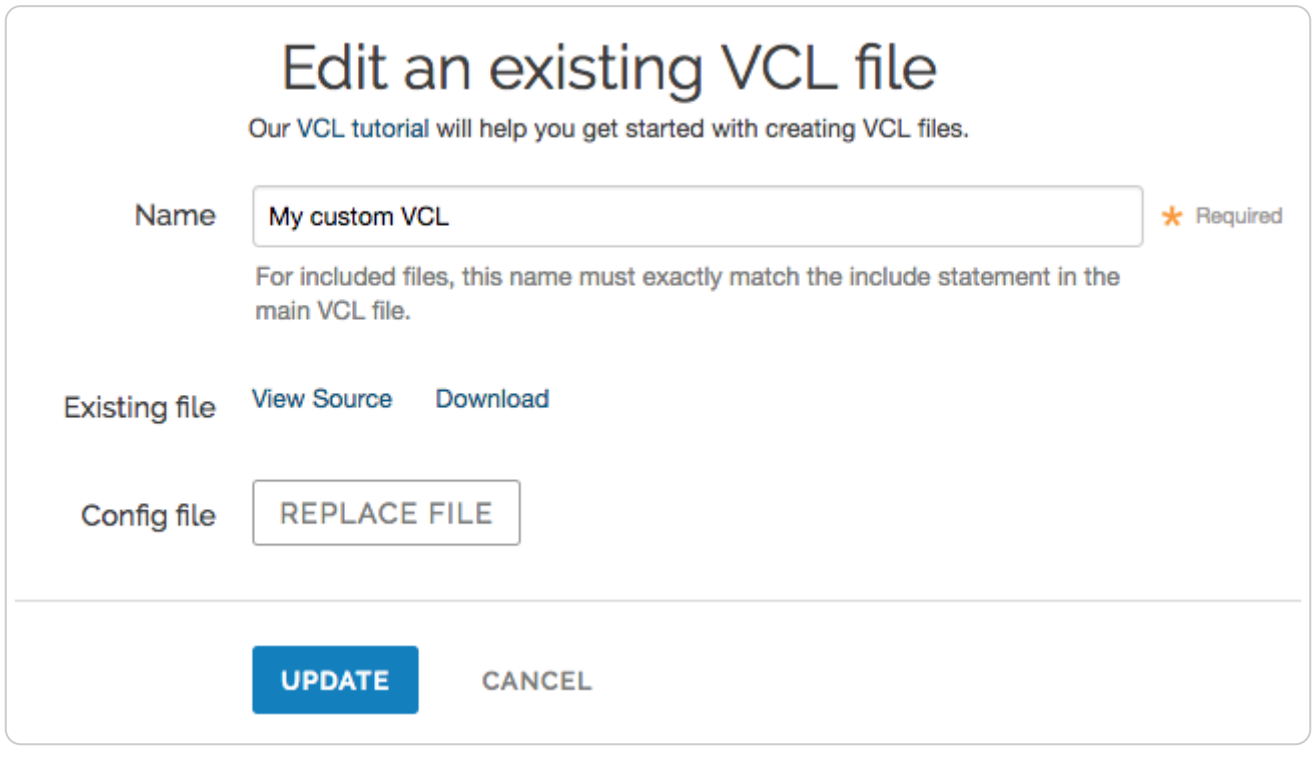
> **IMPORTANT:** Don't upload generated VCL that you've downloaded from the Fastly web interface. Instead, edit and then upload a copy of Fastly's VCL boilerplate to avoid errors.

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 **Edit 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.



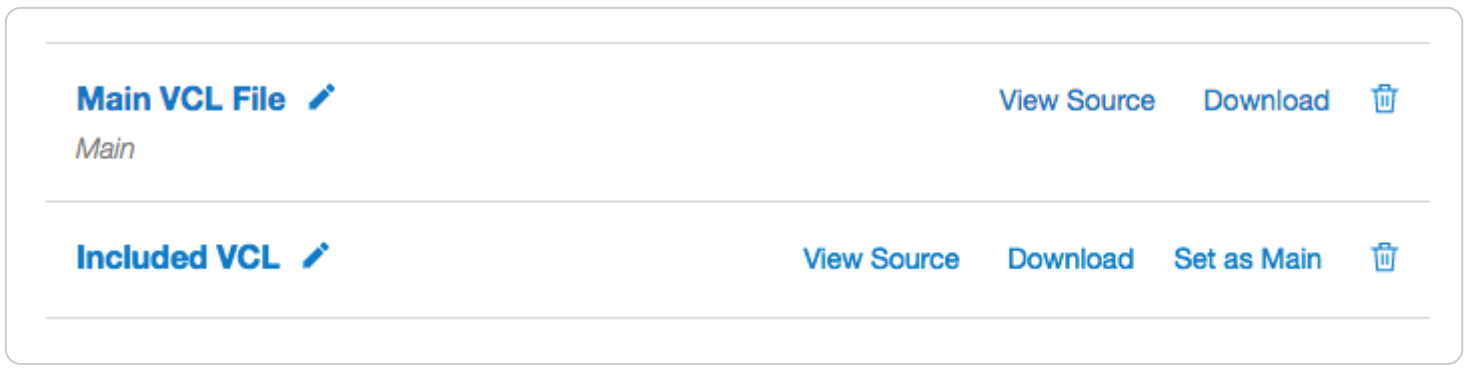
6. In the **Name** field, optionally enter a new name of the VCL file.
7. Click the **Download** link to download the appropriate file.

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

To make your full VCL configuration easier to maintain, you can split it up into multiple files that are accessed by a main VCL file. This allows you to separate out chunks of logic (for example, logic that has a specific purpose or that might change frequently) into as many separate files as makes sense.

1. Start by isolating a portion of VCL and placing it in a separate file. The name of the file doesn't matter, nor does the file extension. A `foo.vcl` file will work just as well as a `bar.txt` file.
2. Upload the file to include it in your Varnish configurations and give it a unique name when you fill out the **Name** field at the time of upload (for example, you could call it `Included VCL`). The uploaded file will appear in the Varnish Configurations area along with your main VCL file.



3. Enter the name of the included VCL file on a separate line in the main VCL configuration file. For example, your **Included VCL** file would get added to the main VCL file in a single line like this:

```
include "Included VCL";
```

4. Continue uploading VCL files and then including them in your main VCL using the syntax `include "<VCL FILE>";` where `<VCL FILE>` exactly matches the name you entered in the **Name** field.

> **TIP:** Our guide to manually creating access control lists demonstrates a common example of using included VCL.

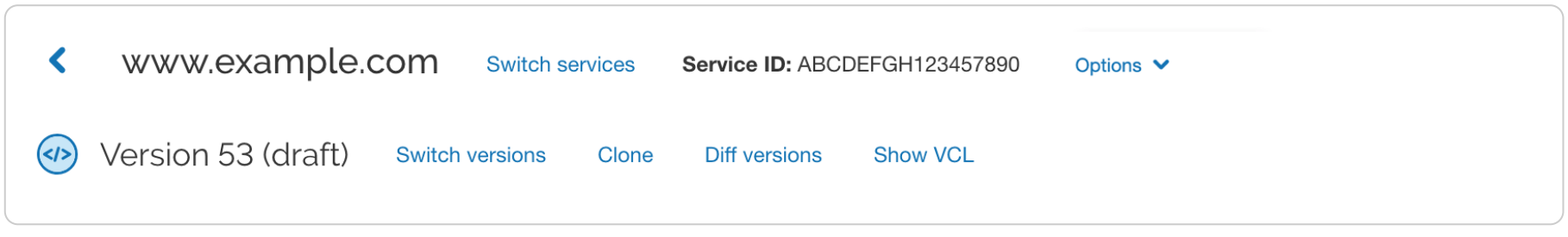
Previewing and testing VCL

Any time you upload VCL files 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 **Edit configuration** button and then select **Clone active**. The Domains page appears.
4. Click the **Show VCL** link.



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 to the production service you'll be using.

§ VCL Snippets

About VCL Snippets

VCL Snippets are short blocks of VCL logic 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. Fastly supports two types of VCL Snippets:

- **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** can be modified and deployed any time they're changed. Because they are versionless objects (much like Edge Dictionaries or ACLs 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.
- Snippets don’t currently support conditions created through the web interface. You can, however, use `if` statements in snippet code.
- Snippets cannot currently be shared between services.

Using dynamic VCL Snippets

Dynamic VCL Snippets are one of two types of snippets that allow you to insert small sections of VCL logic into your service configuration without requiring 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 or ACLs 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 #>/snippet -H "Fastly-Key:FASTL
Y_API_TOKEN" -H 'Content-Type: application/x-www-form-urlencoded' --data $'name=my_dynamic_snippet_name&type=recv&dyn
amic=1&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_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.

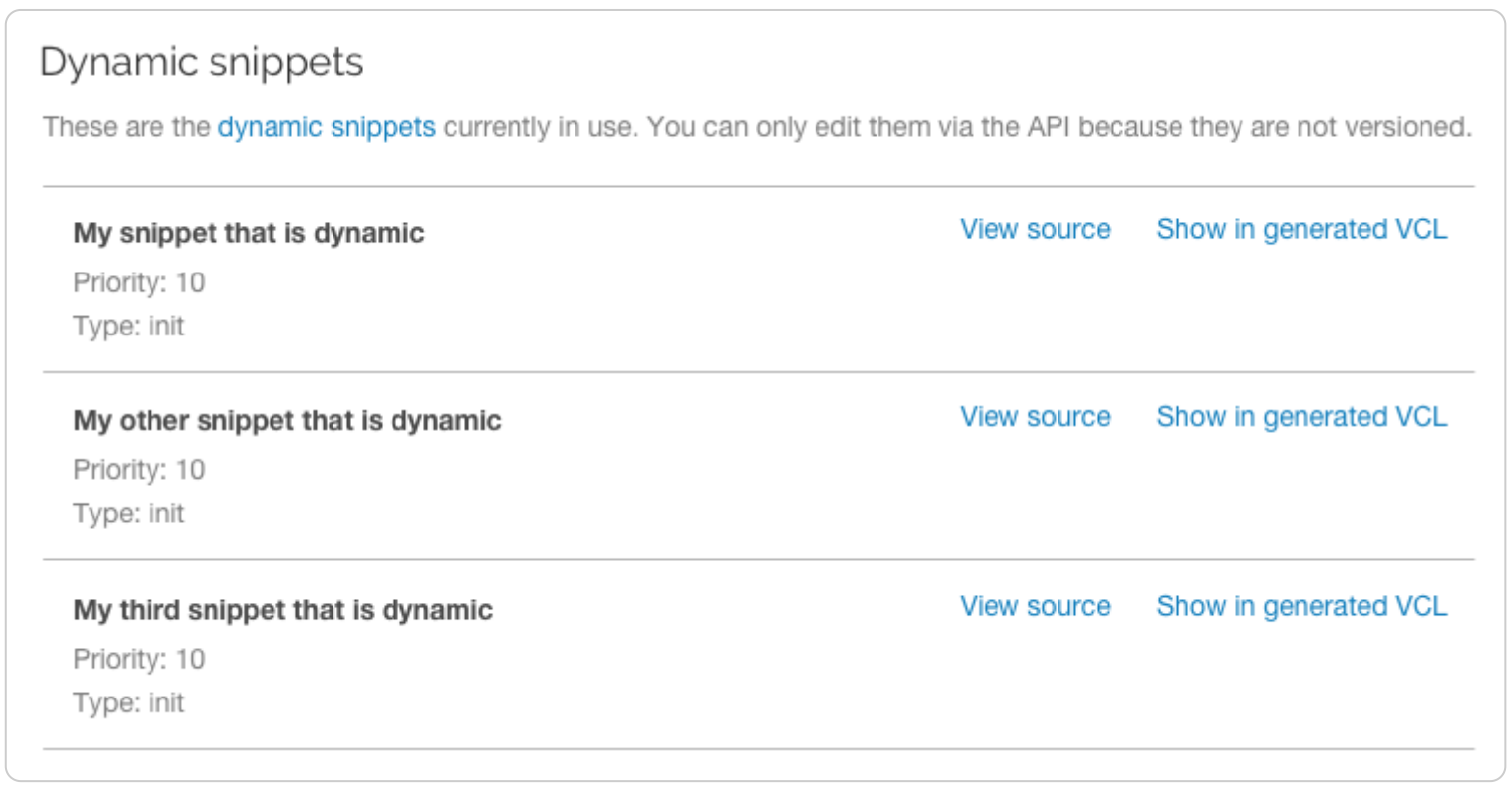
Viewing dynamic VCL Snippets in the web interface

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, 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 $'content=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 #>/snippet/<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

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

Unlike dynamic 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.

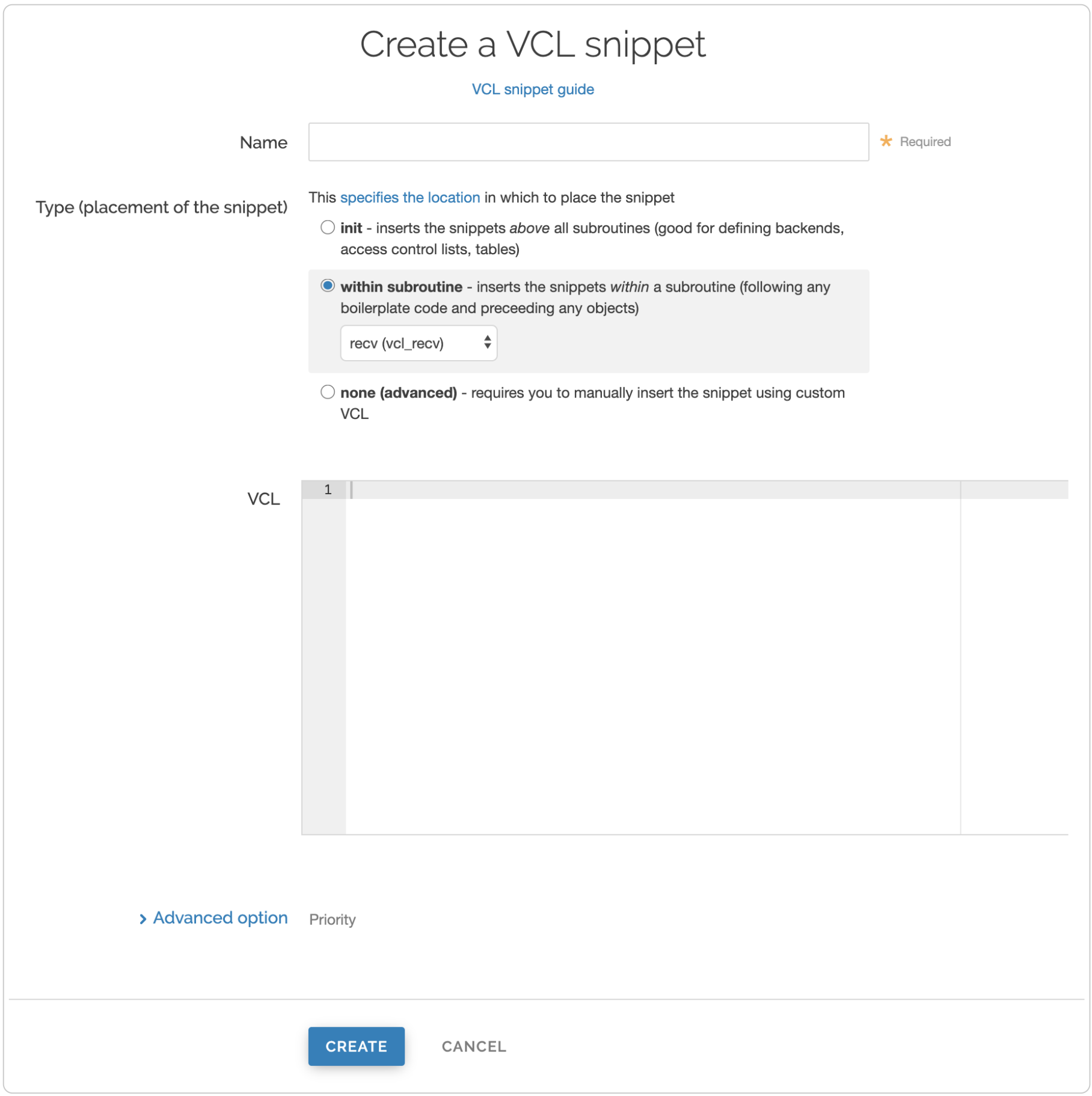
Creating a regular VCL Snippet

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 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 #>/snippet -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. In addition, the returned JSON includes a populated `content` field because the snippet content is stored in a versioned object.

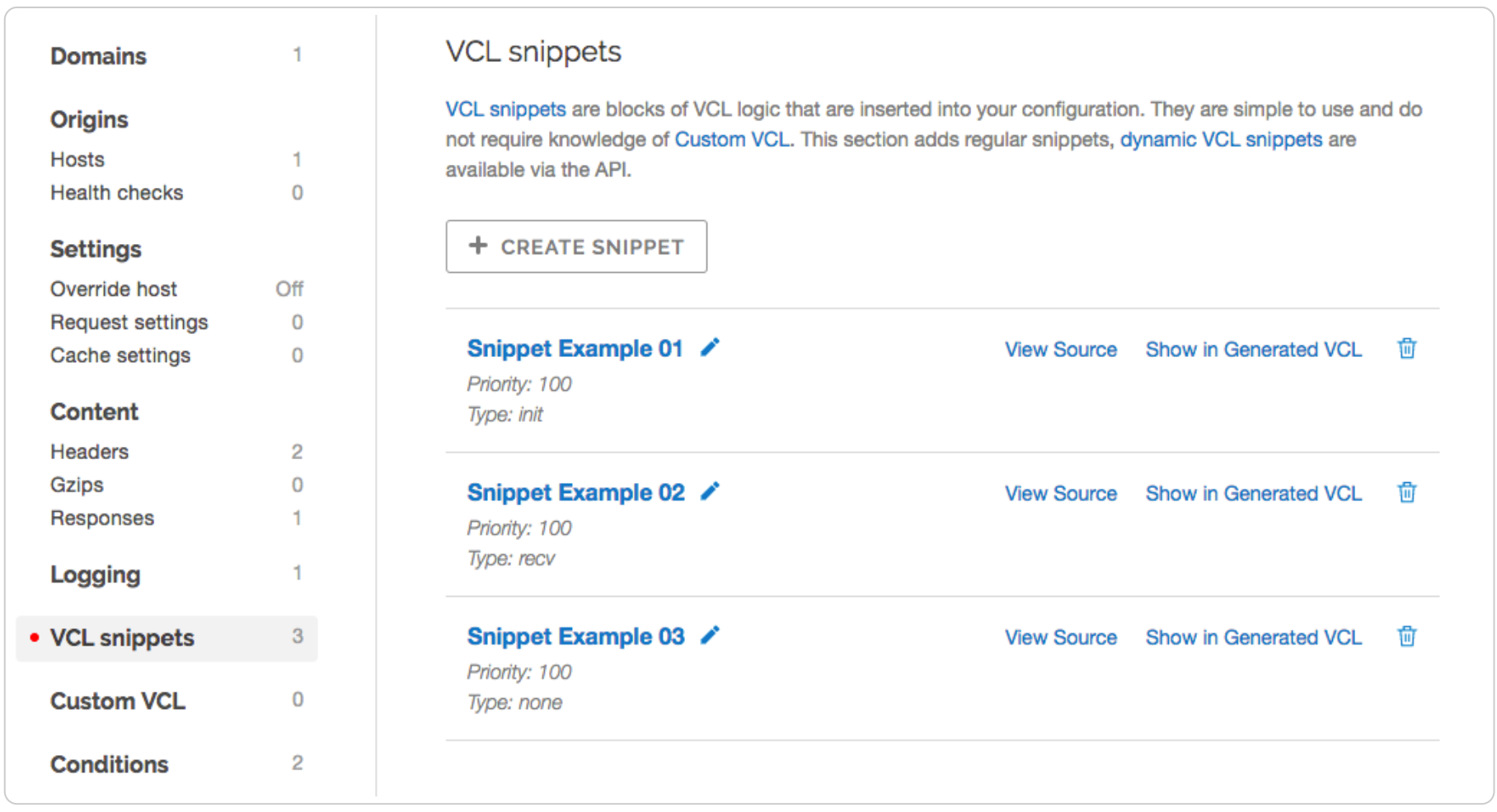
Viewing regular VCL Snippets in the web interface

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 #>/snippet/<Snippet Name e.g my_
regular_snippet> -H "Fastly-Key:FASTLY_API_TOKEN"
```

Unlike fetching dynamic VCL Snippets 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 #>/snippet/ -H "Fastly-Key:FASTL
Y_API_TOKEN"
```

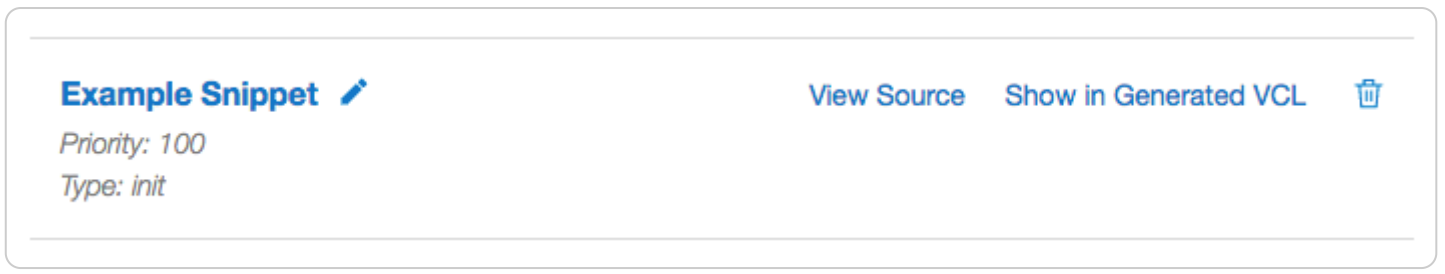
Updating an existing regular VCL Snippet

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

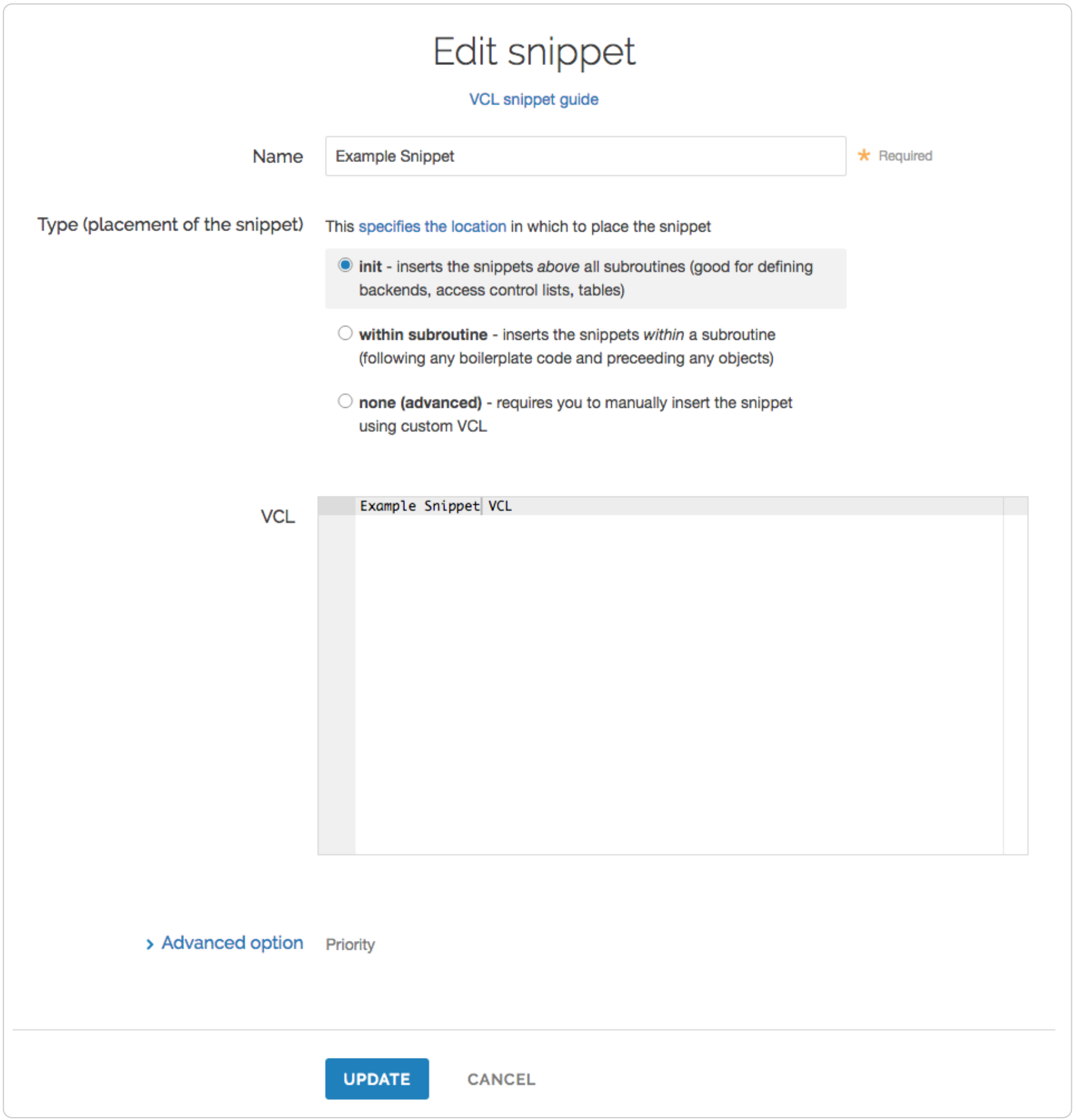
Via the web interface

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:

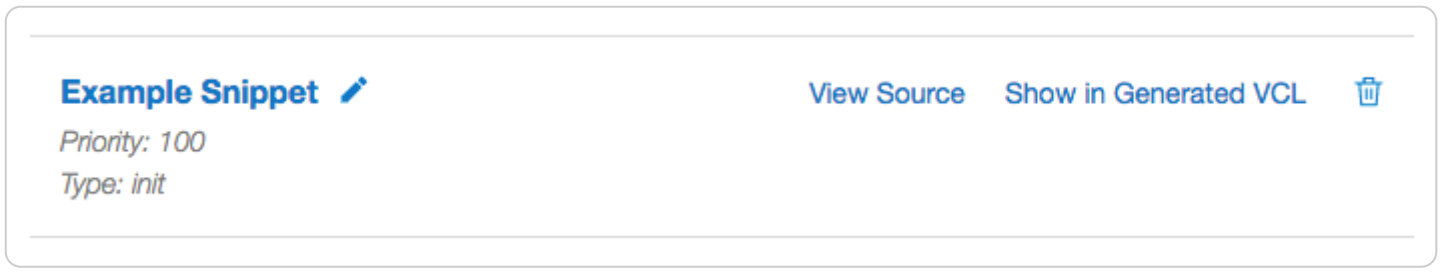
```
curl -X PUT -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippet/<Snippet Name e.g my_
regular_snippet> -H "Fastly-Key:FASTLY_API_TOKEN" -H 'Content-Type: application/x-www-form-urlencoded' --data $'conte
nt=if ( req.url ) {\n set req.http.my-snippet-test-header = \"affirmative\";\n}';
```

Deleting an existing regular VCL Snippet

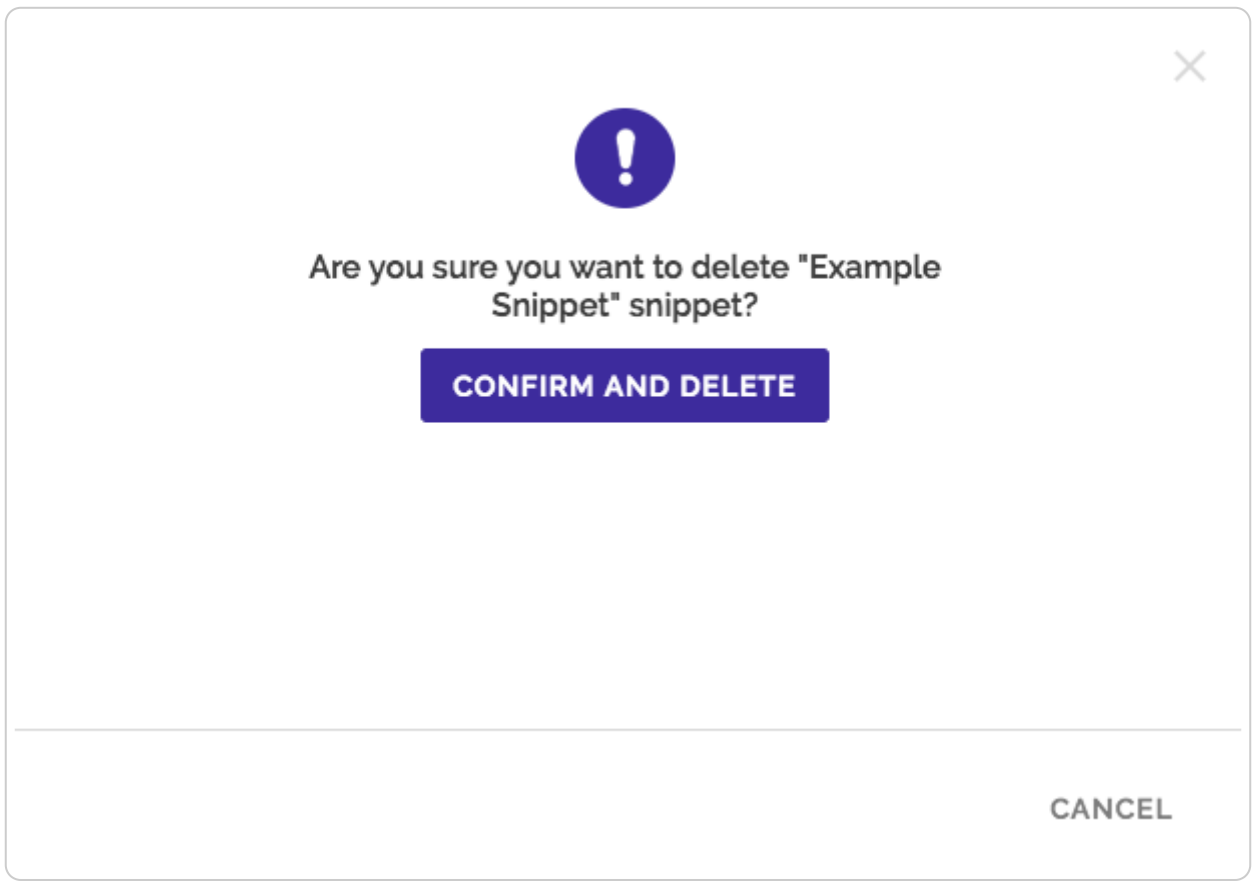
You can update existing regular VCL Snippets via the web interface or via the API.

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 #>/snippet/<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:

```
1  if (req.http.Cookie:edition == "US" || client.geo.country_code == "US" || ) {
2    set req.http.Edition = "US";
3    set req.backend = F_US;
4  } elseif (req.http.Cookie:edition == "Europe" || server.region ~ "^EU-" ) {
5    set req.http.Edition = "EU";
6    set req.backend = F_European;
7  } else {
8    set req.http.Edition = "INT";
9    set req.backend = F_International;
10 }
```

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 and then add a false condition (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

These VCL functions are supported by Fastly.

Content negotiation

Functions for selecting a response from common content negotiation request headers.

- accept.charset_lookup() — Selects the best match from a string in the format of an `Accept-Charset` header's value in the listed character sets, using the algorithm described in Section 5.3.3 of RFC 7231.
- accept.encoding_lookup() — Selects the best match from a string in the format of an `Accept-Encoding` header's value in the listed content encodings, using the algorithm described in Section 5.3.3 of RFC 7231.
- accept.language_filter_basic() — Similar to `accept.language_lookup()`, this function selects the best matches from a string in the format of an `Accept-Language` header's value in the listed languages, using the algorithm described in RFC 4647, Section 3.3.1.
- accept.language_lookup() — Selects the best match from a string in the format of an `Accept-Language` header's value in the listed languages, using the algorithm described in RFC 4647, Section 3.4.
- accept.media_lookup() — Selects the best match from a string in the format of an `Accept` header's value in the listed media types, using the algorithm described in Section 5.3.2 of RFC 7231.

Cryptographic

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

- digest.awsv4_hmac() — Returns an AWSv4 message authentication code based on the supplied `key` and `string`.
- digest.base64_decode() — Returns the Base64 decoding of the input string, as specified by RFC 4648.
- digest.base64() — Returns the Base64 encoding of the input string, as specified by RFC 4648.
- digest.base64url_decode() — Returns the Base64 decoding with URL and filename safe alphabet decoding of the input string, as specified by RFC 4648.
- digest.base64url_nopad_decode() — Returns the Base64 decoding with URL and filename safe alphabet decoding of the input string, as specified by RFC 4648, without padding (`=`).
- digest.base64url_nopad() — Returns the Base64 encoding with URL and filename safe alphabet encoding of the input string, as specified by RFC 4648, without padding (`=`).
- digest.base64url() — Returns the Base64 encoding with URL and filename safe alphabet of the input string, as specified by RFC 4648.
- digest.hash_crc32() — Calculates the 32-bit Cyclic Redundancy Checksum with reversed bit ordering of a string, like that used by bzip2.
- digest.hash_crc32b() — Calculates the 32-bit Cyclic Redundancy Checksum of a string, as specified by ISO/IEC 13239:2002 and section 8.1.1.6.2 of ITU-T recommendation V.42 and used by Ethernet (IEEE 802.3), V.42, FDDI, gzip, zip, and PNG.
- digest.hash_md5() — Use the MD5 hash.
- digest.hash_sha1() — Use the SHA-1 hash.
- digest.hash_sha224() — Use the SHA-224 hash.
- digest.hash_sha256() — Use the SHA-256 hash.
- digest.hash_sha384() — Use the SHA-384 hash.
- digest.hash_sha512() — Use the SHA-512 hash.
- digest.hmac_md5_base64() — Hash-based message authentication code using MD5.
- digest.hmac_md5() — Hash-based message authentication code using MD5.
- digest.hmac_sha1_base64() — Hash-based message authentication code using SHA-1.
- digest.hmac_sha1() — Hash-based message authentication code using SHA-1.
- digest.hmac_sha256_base64() — Hash-based message authentication code using SHA-256.

- digest.hmac_sha256() — Hash-based message authentication code using SHA-256.
- digest.hmac_sha512_base64() — Hash-based message authentication code using SHA-512.
- digest.hmac_sha512() — Hash-based message authentication code using SHA-512.
- digest.rsa_verify() — A boolean function that returns true if the RSA signature of `payload` using `public_key` matches `digest`.
- digest.secure_is_equal() — A boolean function that returns true if s1 and s2 are equal.
- digest.time_hmac_md5() — Returns a time-based one-time password using MD5 based upon the current time.
- digest.time_hmac_sha1() — Returns a time-based one-time password using SHA-1 based upon the current time.
- digest.time_hmac_sha256() — Returns a time-based one-time password with SHA-256 based upon the current time.
- digest.time_hmac_sha512() — Returns a time-based one-time password with SHA-512 based upon the current time.

Date and time

By default VCL includes the `now` variable, which provides the current time (for example, `Mon, 02 Jan 2006 22:04:05 GMT`). Fastly adds several new Varnish variables and functions that allow more flexibility when dealing with dates and times.

- parse_time_delta() — Parses a string representing a time delta and returns an integer number of seconds.
- std.integer2time() — Converts an integer, representing seconds since the UNIX Epoch, to a time variable.
- std.time() — Converts a string to a time variable.
- strftime() — Formats a time to a string.
- time.add() — Adds a relative time to a time.
- time.hex_to_time() — This specialized function takes a hexadecimal string value, divides by `divisor` and interprets the result as seconds since the UNIX Epoch.
- time.is_after() — Returns true if `t1` is after `t2`.
- time.sub() — Subtracts a relative time from a time.

Floating point classification

Floating point classification functions.

- math.is_finite() — Determines whether a floating point value is finite.
- math.is_infinite() — Determines whether a floating point value is an infinity.
- math.is_nan() — Determines whether a floating point value is NaN (Not a Number).
- math.is_normal() — Determines whether a floating point value is normal.
- math.is_subnormal() — Determines whether a floating point value is subnormal.

Math rounding

Rounding of numbers.

- math.ceil() — Computes the smallest integer value greater than or equal to the given value.
- math.floor() — Computes the largest integer value less than or equal to the given value.
- math.round() — Rounds *x* to the nearest integer, with ties away from zero (*commercial rounding*).
- math.roundeven() — Rounds *x* to nearest, ties to even (*bankers' rounding*).
- math.roundhalfdown() — Rounds to nearest, ties towards negative infinity (*half down*).
- math.roundhalfup() — Rounds to nearest, ties towards positive infinity (*half up*).
- math.trunc() — Truncates *x* to an integer value less than or equal in absolute value.

Math trigonometric

Trigonometric functions.

- math.acos() — Computes the principal value of the arc cosine of its argument *x*.
- math.acosh() — Computes the inverse hyperbolic cosine of its argument *x*.
- math.asin() — Computes the principal value of the arc sine of the argument *x*.
- math.asinh() — Computes the inverse hyperbolic sine of its argument *x*.
- math.atan() — Computes the principal value of the arc tangent of its argument *x*.

- math.atan2() — Computes the principal value of the arc tangent of *y/x*, using the signs of both arguments to determine the quadrant of the Return Value.
- math.atanh() — Computes the inverse hyperbolic tangent of its argument *x*.
- math.cos() — Computes the cosine of its argument *x*, measured in radians.
- math.cosh() — Computes the hyperbolic cosine of its argument *x*.
- math.sin() — Computes the sine of its argument *x*, measured in radians.
- math.sinh() — Computes the hyperbolic sine of its argument *x*.
- math.sqrt() — Computes the square root of its argument *x*.
- math.tan() — Computes the tangent of its argument *x*, measured in radians.
- math.tanh() — Computes the hyperbolic tangent of its argument *x*.

Miscellaneous

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

- addr.extract_bits() — Extracts `bit_count` bits (at most 32) starting with the bit number `start_bit` from the given IPv4 or IPv6 address and return them in the form of a non-negative integer.
- addr.is_ipv4() — Returns true if the address family of the given address is IPv4.
- addr.is_ipv6() — Returns true if the address family of the given address is IPv6.
- http_status_matches() — Determines whether the HTTP status matches or does not match any of the statuses in the supplied *fmt* string.
- if() — Implements a ternary operator for strings; if the expression is true, it returns `value-when-true`; if the expression is false, it returns `value-when-false`.
- setcookie.get_value_by_name() — Returns a value associated with the `cookie_name` in the `Set-Cookie` header contained in the HTTP response indicated by `where`.
- std.collect() — Combines multiple instances of the same header into one.
- subfield() — Provides a means to access subfields from a header like `Cache-Control`, `Cookie`, and `Edge-Control` or individual parameters from the query string.

Query string manipulation

Fastly provides a number of extensions to VCL, including several functions for query-string manipulation based on Dridi Boukelmoune's vmod-querystring for Varnish.

- boltsort.sort() — Alias of `querystring.sort`.
- querystring.add() — Returns the given URL with the given parameter name and value appended to the end of the query string.
- querystring.clean() — Returns the given URL without empty parameters.
- querystring.filter_except() — Returns the given URL but only keeps the listed parameters.
- querystring.filter() — Returns the given URL without the listed parameters.
- querystring.filtersep() — Returns the separator needed by the `querystring.filter()` and `querystring.filter_except()` functions.
- querystring.globfilter_except() — Returns the given URL but only keeps the parameters matching a glob.
- querystring.globfilter() — Returns the given URL without the parameters matching a glob.
- querystring.regfilter_except() — Returns the given URL but only keeps the parameters matching a regular expression.
- querystring.regfilter() — Returns the given URL without the parameters matching a regular expression.
- querystring.remove() — Returns the given URL with its query-string removed.
- querystring.set() — Returns the given URL with the given parameter name set to the given value, replacing the original value and removing any duplicates.
- querystring.sort() — Returns the given URL with its query-string sorted.

Randomness

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

- randombool_seeded() — Identical to randombool, except takes an additional parameter, which is used to seed the random number generator.

- randombool() — Returns a random, boolean value.
- randomint_seeded() — Identical to randomint, except takes an additional parameter used to seed the random number generator.
- randomint() — Returns a random integer value between `from` and `to`, inclusive.
- randomstr() — Returns a random string of length `len` containing characters from the supplied string `characters`.

String manipulation

These functions provide various manipulation for strings containing arbitrary text content.

- cstr_escape() — Escapes bytes from a string using C-style escape sequences.
- json.escape() — Escapes characters of a UTF-8 encoded Unicode string using JSON-style escape sequences.
- regsub() — Replaces the first occurrence of `pattern`, which is a Perl-compatible regular expression, in `input` with `replacement`.
- regsuball() — Replaces all occurrences of `pattern`, which may be a Perl-compatible regular expression, in `input` with `replacement`.
- std.anystr2ip() — Converts the string `addr` to an IP address (IPv4 or IPv6).
- std.atof() — Takes a string (which represents a float) as an argument and returns its value.
- std.atoi() — Takes a string (which represents an integer) as an argument and returns its value.
- std.ip() — An alias of `std.str2ip()`.
- std.ip2str() — Converts the IP address (v4 or v6) to a string.
- std.prefixof() — True if the string `s` begins with the string `begins_with`.
- std.str2ip() — Converts the string representation of an IP address (IPv4 or IPv6) into an `IP type`.
- std.strlen() — Returns the length of the string.
- std.strpad() — This function constructs a string containing the input string `s` padded out with `pad` to produce a string of the given `width`.
- std.strrep() — Repeats the given string `n` times.
- std.strrev() — Reverses the given string.
- std.strstr() — Returns the part of `haystack` string starting from and including the first occurrence of `needle` until the end of `haystack`.
- std.strtof() — Converts the string *s* to a float value with the given base *base*.
- std.strtol() — Converts the string *s* to an integer value.
- std.suffixof() — True if the string `s` ends with the string `ends_with`.
- std.tolower() — Changes the case of a string to lowercase.
- std.toupper() — Changes the case of a string to upper case.
- substr() — Returns a substring of the byte string *s*, starting from the byte *offset*, of byte *length*.
- urldecode() — Decodes a percent-encoded string.
- urlencode() — Encodes a string for use in a URL.
- utf8.codepoint_count() — Returns the number of UTF-8 encoded Unicode code points in the string *s*.
- utf8.is_valid() — Returns true if the string *s* contains valid UTF-8 and returns false if it does not contain valid UTF-8.
- utf8.strpad() — Like `std.strpad()` except `count` gives the number of unicode code points for the output string rather than bytes.
- utf8.substr() — Returns a substring of the UTF-8 string *s*, starting from the Unicode code point *offset*, of Unicode code point *length*.

Table

Tables provide a means to declare a constant dictionary and to efficiently look up values in the dictionary.

- table.lookup() — Look up the key `key` in the table `ID`.

TLS and HTTP/2

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

- h2.disable_header_compression() — Sets a flag to disable HTTP/2 header compression on one or many response headers to the client.
- h2.push() — Triggers an HTTP/2 server push of the asset passed into the function as the input-string.

UUID

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

- uuid.dns() — Returns the RFC4122 identifier of DNS namespace, namely the constant `"6ba7b810-9dad-11d1-80b4-00c04fd430c8"`.
- uuid.is_valid() — Returns true if the string holds a textual representation of a valid UUID (per RFC4122).
- uuid.is_version3() — Returns true if string holds a textual representation of a valid version 3 UUID.
- uuid.is_version4() — Returns true if string holds a textual representation of a valid version 4 UUID.
- uuid.is_version5() — Returns true if string holds a textual representation of a valid version 5 UUID.
- uuid.oid() — Returns the RFC4122 identifier of ISO OID namespace, namely the constant `"6ba7b812-9dad-11d1-80b4-00c04fd430c8"`.
- uuid.url() — Returns the RFC4122 identifier of URL namespace, namely the constant `"6ba7b811-9dad-11d1-80b4-00c04fd430c8"`.
- uuid.version3() — Derives a UUID corresponding to `name` within the given `namespace` using MD5 hash function.
- uuid.version4() — Returns a UUID based on random number generator output.
- uuid.version5() — Derives a UUID corresponding to `name` within the given `namespace` using SHA-1 hash function.
- uuid.x500() — Returns the RFC4122 identifier of X.500 namespace, namely the constant `"6ba7b812-9dad-11d1-80b4-00c04fd430c8"`.

Variables

These VCL variables are supported by Fastly.

Date and time

By default VCL includes the `now` variable, which provides the current time (for example, `Mon, 02 Jan 2006 22:04:05 GMT`). Fastly adds several new Varnish variables and functions that allow more flexibility when dealing with dates and times.

- now.sec — Like the `now` variable, but in seconds since the UNIX Epoch.
- now — The current time in RFC 1123 format (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`).
- time.elapsed.msec_frac — The time that has elapsed in milliseconds since the request started.
- time.elapsed.msec — The time since the request start in milliseconds.
- time.elapsed.sec — The time since the request start in seconds.
- time.elapsed.usec_frac — The time the request started in microseconds since the last whole second.
- time.elapsed.usec — The time since the request start in microseconds.
- time.elapsed — The time since the request started.
- time.end.msec_frac — The time the request started in milliseconds since the last whole second.
- time.end.msec — The time the request ended in milliseconds since the UNIX Epoch.
- time.end.sec — The time the request ended in seconds since the UNIX Epoch.
- time.end.usec_frac — The time the request started in microseconds since the last whole second.
- time.end.usec — The time the request ended in microseconds since the UNIX Epoch.
- time.end — The time the request ended, using RFC 1123 format (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`).
- time.start.msec_frac — The time the request started in milliseconds since the last whole second, after TLS termination.
- time.start.msec — The time the request started in milliseconds since the UNIX Epoch, after TLS termination.
- time.start.sec — The time the request started in seconds since the UNIX Epoch, after TLS termination.
- time.start.usec_frac — The time the request started in microseconds since the last whole second, after TLS termination.

- time.start.usec — The time the request started in microseconds since the UNIX Epoch, after TLS termination.
- time.start — The time the request started, after TLS termination, using RFC 1123 format (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`).
- time.to_first_byte — The time interval since the request started up to the point before the `vcl_deliver` function ran.

Edge Side Includes (ESI)

Fastly exposes tools to allow you to track a request that has ESI.

- req.esi — Whether or not to disable or enable ESI processing during this request.
- req.topurl — In an ESI subrequest, contains the URL of the top-level request.

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 — The name of the organization associated with `client.as.number`.
- client.as.number — Autonomous system (AS) number.
- client.geo.area_code — The telephone area code associated with the IP address.
- client.geo.city.ascii — City or town name, encoded using ASCII encoding.
- client.geo.city.latin1 — City or town name, encoded using Latin-1 encoding.
- client.geo.city.utf8 — City or town name, encoded using UTF-8 encoding.
- client.geo.city — Alias of `client.geo.city.ascii`.
- client.geo.conn_speed — Connection speed.
- client.geo.continent_code — Two-letter code representing the continent.
- client.geo.country_code — A two-character ISO 3166-1 country code for the country associated with the IP address.
- client.geo.country_code3 — A three-character ISO 3166-1 alpha-3 country code for the country associated with the IP address.
- client.geo.country_name.ascii — Country name, encoded using ASCII encoding.
- client.geo.country_name.latin1 — Country name, encoded using Latin-1 encoding.
- client.geo.country_name.utf8 — Country name, encoded using UTF-8 encoding.
- client.geo.country_name — Alias of `client.geo.country_name.ascii`.
- client.geo.gmt_offset — Time zone offset from coordinated universal time (UTC) for `client.geo.city`.
- client.geo.ip_override — Override the IP address for geolocation data.
- client.geo.latitude — Latitude, in units of degrees from the equator.
- client.geo.longitude — Longitude, in units of degrees from the IERS Reference Meridian.
- client.geo.metro_code — Metro code.
- client.geo.postal_code — The postal code associated with the IP address.
- client.geo.region.ascii — ISO 3166-2 country subdivision code.
- client.geo.region.latin1 — Region code, encoded using Latin-1 encoding.
- client.geo.region.utf8 — Region code, encoded using UTF-8 encoding.
- client.geo.region — Alias of `client.geo.region.ascii`.

Math constants and limits

Features that support various math constants and limits.

- math.1_PI — The value of the reciprocal of `math.PI` (1/Pi).
- math.2_PI — The value of two times the reciprocal of `math.PI` (2/Pi).
- math.2_SQRTPI — The value of two times the reciprocal of the square root of `math.PI` (2/sqrt(Pi)).
- math.2PI — The value of `math.PI` multiplied by two (Tau).
- math.E — The value of the base of natural logarithms (e).
- math.FLOAT_DIG — Number of decimal digits that can be stored without loss in the `FLOAT` type.
- math.FLOAT_EPSILON — Minimum positive difference from 1.0 for the `FLOAT` type.

- math.FLOAT_MANT_DIG — Number of hexadecimal digits stored for the significand in the `FLOAT` type.
- math.FLOAT_MAX_10_EXP — Maximum value in base 10 of the exponent part of the `FLOAT` type.
- math.FLOAT_MAX_EXP — Maximum value in base 2 of the exponent part of the `FLOAT` type.
- math.FLOAT_MAX — Maximum finite value for the `FLOAT` type.
- math.FLOAT_MIN_10_EXP — Minimum value in base 10 of the exponent part of the `FLOAT` type.
- math.FLOAT_MIN_EXP — Minimum value in base 2 of the exponent part of the `FLOAT` type.
- math.FLOAT_MIN — Minimum finite value for the `FLOAT` type.
- math.INTEGER_BIT — Number of bits in the `INTEGER` type.
- math.INTEGER_MAX — Maximum value for the `INTEGER` type.
- math.INTEGER_MIN — Minimum value for the `INTEGER` type.
- math.LN10 — The value of the natural logarithm of 10 (log_e 10).
- math.LN2 — The value of the natural logarithm of 2 (log_e 2).
- math.LOG10E — The value of the logarithm to base 10 of `math.E` (log_10 e).
- math.LOG2E — The value of the logarithm to base 2 of `math.E` (log_2 e).
- math.NAN — A value that is "not a number." When converted to a STRING value, this is rendered as `NaN`.
- math.NEG_HUGE_VAL — Negative overflow value.
- math.NEG_INFINITY — A value representing negative infinity (−∞).
- math.PHI — The golden ratio (Φ).
- math.PI_2 — The value of `math.PI` divided by two (Pi/2).
- math.PI_4 — The value of `math.PI` divided by four (Pi/4).
- math.PI — The value of the ratio of a circle's circumference to its diameter (Pi).
- math.POS_HUGE_VAL — Positive overflow value.
- math.POS_INFINITY — A value representing positive infinity (+∞).
- math.SQRT1_2 — The value of the reciprocal of the square root of two (1/sqrt(2)).
- math.SQRT2 — The value of the square root of two (sqrt(2)).
- math.TAU — The value of `math.PI` multiplied by two (Tau).

Miscellaneous

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

- bereq.url.basename — Same as `req.url.basename`, except for use between Fastly and your origin servers.
- bereq.url.dirname — Same as `req.url.dirname`, except for use between Fastly and your origin servers.
- bereq.url.qs — The query string portion of `bereq.url`.
- bereq.url — The URL sent to the backend.
- beresp.backend.ip — The IP of the backend this response was fetched from (backported from Varnish 3).
- beresp.backend.name — The name of the backend this response was fetched from (backported from Varnish 3).
- beresp.backend.port — The port of the backend this response was fetched from (backported from Varnish 3).
- beresp.grace — Defines how long an object can remain overdue and still have Varnish consider it for grace mode.
- beresp.hipaa — Specifies that content not be cached in non-volatile memory to help customers meet HIPAA security requirements.
- beresp.pci — Specifies that content be cached in a manner that satisfies PCI DSS requirements.
- client.ip — The IP address of the client making the request.
- client.port — Returns the remote client port.
- client.requests — Tracks the number of requests received by Varnish over a persistent connection.
- client.socket.pace — Ceiling rate in kilobytes per second for bytes sent to the client.
- fastly.error — Contains the error code raised by the last function, otherwise *not set*.
- req.backend.healthy — Whether or not this backend, or recursively any of the backends under this director, is considered healthy.

- req.backend.is_cluster — True if this backend, or recursively any of the backends under this director, is a cluster backend.
- req.backend.is_origin — True if this backend, or recursively any of the backends under this director, is not a shield backend.
- req.backend.is_shield — True if this backend, or recursively any of the backends under this director, is a shield backend.
- req.backend — The backend to use to service the request.
- req.body.base64 — Same as `req.body`, except the request body is encoded in Base64, which handles null characters and allows representation of binary bodies.
- req.body — The request body.
- req.grace — Defines how long an object can remain overdue and still have Varnish consider it for grace mode.
- req.http.host — The full host name, without the path or query parameters.
- req.is_ipv6 — Indicates whether the request was made using IPv6 or not.
- req.restarts — Counts the number of times the VCL has been restarted.
- req.url.basename — The file name specified in a URL.
- req.url.dirname — The directories specified in a URL.
- req.url.ext — The file extension specified in a URL.
- req.url.path — The full path, without any query parameters.
- req.url.qs — The query string portion of `req.url`.
- req.url — The full path, including query parameters.
- stale.exists — Specifies if a given object has stale content in cache.

Segmented Caching

Variables related to controlling range requests via Segmented Caching.

- fastly.segmented_caching.autopurged — Whether an inconsistency encountered during Segmented Caching processing led to the system automatically enqueuing a purge request.
- fastly.segmented_caching.block_number — A zero-based counter identifying the file fragment being processed.
- fastly.segmented_caching.cancelled — Whether Segmented Caching processing was enabled and cancelled by a non-206 response.
- fastly.segmented_caching.client_req.is_open_ended — Whether the client's request leaves the upper bound of the range open.
- fastly.segmented_caching.client_req.is_range — Whether the client's request is a range request.
- fastly.segmented_caching.client_req.range_high — The upper bound of the client's requested range.
- fastly.segmented_caching.client_req.range_low — The lower bound of the client's requested range.
- fastly.segmented_caching.completed — Whether Segmented Caching processing was enabled and cancelled by a non-206 response.
- fastly.segmented_caching.error — The reason why Segmented Caching processing failed.
- fastly.segmented_caching.failed — Whether Segmented Caching processing was enabled and ended in a failure.
- fastly.segmented_caching.is_inner_req — Whether VCL is running in the context of a sub-request that is retrieving a fragment of a file.
- fastly.segmented_caching.is_outer_req — Whether VCL is running in the context of a request that is assembling file fragments into a response.
- fastly.segmented_caching.obj.complete_length — The size of the whole file in bytes.
- fastly.segmented_caching.rounded_req.range_high — The upper bound of the rounded block being processed.
- fastly.segmented_caching.rounded_req.range_low — The lower bound of the rounded block being processed.
- fastly.segmented_caching.total_blocks — The number of fragments needed for assembling this response.

Server

Variables relating to the server receiving the request.

- server.datacenter — A code representing one of Fastly's POP locations.
- server.hostname — Hostname of the server (e.g., `cache-jfk1034`).
- server.identity — Same as `server.hostname` but also explicitly includes the datacenter name (e.g., `cache-jfk1034-JFK`).

- server.region — A code representing the general region of the world in which the POP location resides.

Size

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

- bereq.body_bytes_written — Total body bytes written to a backend.
- bereq.header_bytes_written — Total header bytes written to a backend.
- req.body_bytes_read — Total body bytes read from the client generating the request.
- req.bytes_read — Total bytes read from the client generating the request.
- req.header_bytes_read — Total header bytes read from the client generating the request.
- resp.body_bytes_written — Body bytes to send to the client in the response.
- resp.bytes_written — Total bytes to send to the client in the response.
- resp.completed — Whether the response completed successfully or not.
- resp.header_bytes_written — How many bytes were written for the header of a response.

TLS and HTTP/2

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

- fastly_info.h2.is_push — Whether or not this request was a server-initiated request generated to create an HTTP/2 Server-pushed response.
- fastly_info.h2.stream_id — If the request was made over HTTP/2, the underlying HTTP/2 stream ID.
- fastly_info.is_h2 — Whether or not the request was made using HTTP/2.
- tls.client.cipher — The cipher suite used to secure the client TLS connection.
- tls.client.ciphers_list_sha — A SHA-1 digest of the raw buffer containing the list of supported ciphers, represented in Base64.
- tls.client.ciphers_list_txt — The list of ciphers supported by the client, rendered as text, in a colon-separated list.
- tls.client.ciphers_list — The list of ciphers supported by the client, as sent over the network, hex encoded.
- tls.client.ciphers_sha — A SHA-1 of the cipher suite identifiers sent from the client as part of the TLS handshake, represented in Base64.
- tls.client.protocol — The TLS protocol version this connection is speaking over.
- tls.client.servername — The Server Name Indication (SNI) the client sent in the `ClientHello` TLS record.
- tls.client.tlsexts_list_sha — A SHA-1 digest of the TLS extensions supported by the client as little-endian, 16-bit integers, represented in Base64.
- tls.client.tlsexts_list_txt — The list of TLS extensions supported by the client, rendered as text in a colon-separated list.
- tls.client.tlsexts_list — The list of TLS extensions supported by the client as little-endian, 16-bit, unsigned integers, hex encoded.
- tls.client.tlsexts_sha — A SHA-1 of the TLS extension identifiers sent from the client as part of the TLS handshake, represented in Base64.

Local variables

Fastly VCL supports variables for storing temporary values during request processing.

> ★ **TIP:** Consider using a `req.http.*` header to store a value if you need to pass information between functions or to the origin.

Declaring a variable

Variables must be declared before they are used, usually at the beginning of a function before any statements. They can only be used in the same function where they are declared. Fastly VCL does not provide block scope. Declarations apply to an entire function's scope even if a variable is declared within a block.

Variables start with `var.` and their names consist of characters in the set `[A-Za-z0-9._-]`. (`:` is explicitly disallowed.) The declaration syntax is:

```
declare local var.<name> <type>;
```

Variable types

Variables can be of the following types:

- `BOOL`
- `FLOAT`
- `INTEGER`
- `IP`
- `RTIME` (relative time)
- `STRING`
- `TIME` (absolute time)

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.";
```

IP address variables

IP address variables represent individual IP addresses.

```
acl office_ip_ranges {
  "192.0.2.0"/24;                              # internal office
  "198.51.100.4";                              # remote VPN office
  "2001:db8:ffff:ffff:ffff:ffff:ffff:ffff";    # ipv6 address remote
}

declare local var.ip1 IP;
set var.ip1 = "192.0.2.0";

if (var.ip1 ~ office_ip_ranges) {
  ...
}

declare local var.ip2 IP;
set var.ip2 = "2001:db8:ffff:ffff:ffff:ffff:ffff:ffff";
```

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

Fastly 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.

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

String concatenation

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

```
1   := string-expr string-expr
2    | 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:

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

Multiplication, division and modulus

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

```
1   := "set" variable "*=" assignment-expr ";"
2    | "set" variable "/=" assignment-expr ";"
3    | "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: $-\infty < n < +\infty$

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:

```
1  := string-expr re-op STRING
2   | 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
`;`	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 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
- FLOAT
- INTEGER
- IP
- RTIME
- STRING
- 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
- HASH
- HEADER
- VOID

Directors

Fastly's directors contain a list of backends to direct requests to. Traffic is distributed according to the specific director policy.

Healthcheck probes should be defined for backends within directors so the director can check the backend health state before sending a request. Directors will not send traffic to a backend that is identified as unhealthy.

Random director

The random director selects a backend randomly from the healthy subset of backends.

Each backend has a `.weight` attribute that indicates the weighted probability of the director selecting the backend.

The random director has the following properties:

- `retries`: The number of times the director will try to find a healthy backend or connect to the randomly chosen backend if the first connection attempt fails. If `.retries` is not specified, then the director will use the number of backend members as the retry limit.
- `quorum`: The percentage threshold that must be reached by the cumulative `.weight` of all healthy backends in order for the director to be deemed healthy. If `.quorum` is not specified, the director will use 0 as the quorum weight threshold.

In the following example, the random director will randomly select a backend with equal probability. At minimum, two backends must be healthy for their cumulative weight (~ 66%) to exceed the 50% quorum weight and qualify the director as healthy. If only one backend is healthy and the quorum weight is not reached, a "Quorum weight not reached" error will be returned to the client. If the random director fails to connect to the chosen backend, it will retry randomly selecting a backend up to three times before indicating all backends are unhealthy.

```
director my_dir random {
    .quorum = 50%;
    .retries = 3;
    { .backend = F_backend1; .weight = 1; }
    { .backend = F_backend2; .weight = 1; }
    { .backend = F_backend3; .weight = 1; }
}
```

Round-robin director

WARNING: Fastly plans to retire the Round-robin director on May 8, 2020. It will not be supported beyond the retirement date for existing users. Fastly does not recommend new customers use the Round-robin director. Our existing Random director, with its associated API, offers similar functionality to the deprecated director policy.

The round-robin director will send requests in a round-robin fashion to each healthy backend in its backend list.

In the following example, the round-robin director will send its first request to `F_backend1`, second request to `F_backend2`, third request to `F_backend3`, fourth request to `F_backend1`, and so on.

```
director my_dir round-robin {
    { .backend = F_backend1; }
    { .backend = F_backend2; }
    { .backend = F_backend3; }
}
```

Fallback director

The fallback director always selects the first healthy backend in its backend list to send requests to. If Varnish fails to establish a connection with the chosen backend, the director will select the next healthy backend.

In the following example, the fallback director will send requests to `F_backend1` until its health status is unhealthy. If the Varnish client is unable to connect to `F_backend1` (e.g., a 503 connection timed out response is returned), the fallback director will select the next healthy backend. If all backends in the list are unhealthy or all backends fail to accept connections, a 503 all backends failed or unhealthy response is returned to the client.

```
director my_dir fallback {
  { .backend = F_backend1; }
  { .backend = F_backend2; }
  { .backend = F_backend3; }
}
```

Rounding modes

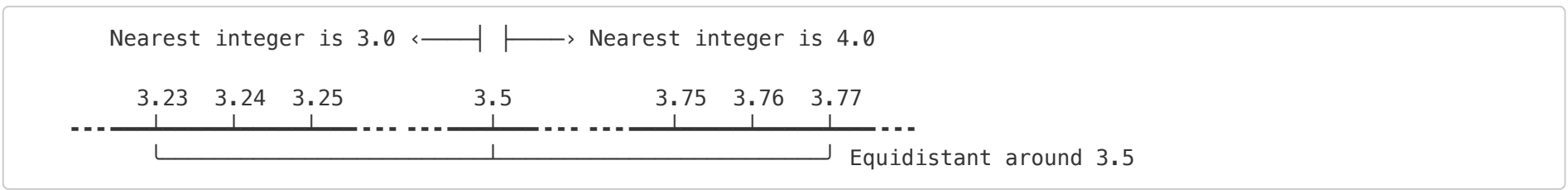
Fastly VCL provides access to various *rounding modes* by way of independent functions for rounding values. These functions have explicit rounding modes. There is no stateful interface to set a "current" rounding mode.

Fastly VCL does not provide interfaces to round values to a given number of significant figures, to a given multiple, or to a given power.

Tie-breaking when rounding to nearest

The roundoff errors introduced by rounding values to their nearest integers are symmetric, except for treatment of the exact midpoint between adjacent integers.

That is, for every value that gets rounded up (such as 3.77 rounding up to the nearest integer 4.0), there is a corresponding value (3.23) which is rounded down by the same amount. This can be seen visually:



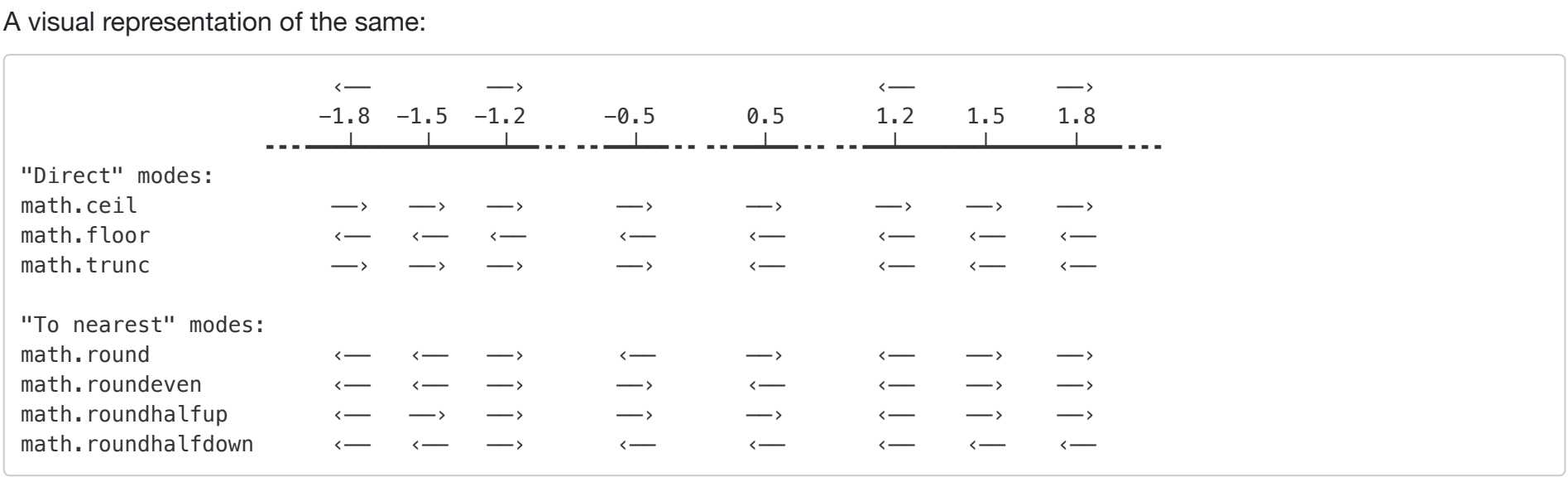
Rounding to the nearest integer requires a tie-breaking rule for when the fractional part of a value is exactly 0.5. There are several ways to break these ties, enumerated in the "to nearest" rounding modes below.

Overview

Example values:

Input	ceil	floor	trunc	round	roundeven	roundhalfup	roundhalfdown
-1.8	-1.0	-2.0	-1.0	-2.0	-2.0	-2.0	-2.0
-1.5	-1.0	-2.0	-1.0	-2.0	-2.0	-1.0	-2.0
-1.2	-1.0	-2.0	-1.0	-1.0	-1.0	-1.0	-1.0
-0.5	-0.0	-1.0	-0.0	-1.0	-0.0	-0.0	-1.0
0.5	1.0	0.0	0.0	1.0	0.0	1.0	0.0
1.2	2.0	1.0	1.0	1.0	1.0	1.0	1.0
1.5	2.0	1.0	1.0	2.0	2.0	2.0	1.0
1.8	2.0	1.0	1.0	2.0	2.0	2.0	2.0

A visual representation of the same:



"Direct" rounding modes

- **Round up** — math.ceil()

  Also known as *ceiling*, *round towards positive infinity*

  IEEE 754 roundTowardPositive

  Non-integer values are rounded up towards +∞. Negative results thus round toward zero.

- **Round down** — math.floor()

Also known as *floor*, *round towards negative infinity*

IEEE 754 roundTowardNegative

Non-integer values are rounded down towards -∞. Negative results thus round away from zero.

- **Round towards zero** — math.trunc()

  Also known as *truncation*, *round away from infinity*

  IEEE 754 roundTowardZero

  Rounding is performed by removing the fractional part of a number, leaving the integral part unchanged.

  > **NOTE:** The `FLOAT` to `INTEGER` type conversion in Fastly VCL is not by truncation (as it is in many comparable languages). See discussion under ties away from zero.

- **Round away from zero**

  Also known as *round towards infinity*

  Positive non-integer values are rounded up towards positive infinity. Negative non-integer values are rounded down towards negative infinity.

  Not provided in Fastly VCL.

"To nearest" rounding modes

All of the following modes round non-tie values to their nearest integer. These modes differ only in their treatment of ties.

- **Round to nearest, ties away from zero** — math.round()

  Also known as *commercial rounding*

  IEEE 754 roundTiesToAway

  For positive values, ties are rounded up towards positive infinity. For negative values, ties are rounded down towards negative infinity.

  This is symmetric behavior, avoiding bias to either positive or negative values. However, this mode does introduce bias away from zero.

  This rounding mode is used for implicit `FLOAT` to `INTEGER` type conversions in VCL. These behave as if by a call to `math.round()`.

- **Round to nearest, ties to even** — math.roundeven()

  Also known as *half to even*, *convergent rounding*, *statistician's rounding*, *Dutch rounding*, *Gaussian rounding*, *odd–even rounding*, and *bankers' rounding*

  IEEE 754 roundTiesToEven

  Of the two nearest integer values, ties are rounded either up or down to whichever value is even.

  This rounding mode increases the probability of even numbers relative to odd numbers, but avoids bias to either positive or negative values, and also avoids bias towards or away from zero. The cumulative error is minimized when summing rounded values, especially when the values are predominantly positive or predominantly negative.

- **Round to nearest, ties towards positive infinity** — math.roundhalfup()

  Also known as *half up*

  This is asymmetric behavior, where ties for negative values are rounded towards zero, and ties for positive values are rounded away from zero.

  > **WARNING:** Some languages use the term *half up* to mean symmetric behavior. For rounding functions in these languages, "up" is a value of larger absolute magnitude. That is, negative ties will be rounded away from zero, which differs from the behavior in VCL. Take care when porting code using this rounding mode to VCL.

- **Round to nearest, ties towards negative infinity** — math.roundhalfdown()

  Also known as *half down*

  This is asymmetric behavior, where ties for negative values are rounded away from zero, and ties for positive values are rounded towards zero.

  > **WARNING:** Some languages use the term *half down* to mean symmetric behavior. For rounding functions in these languages, "down" is a value of smaller absolute magnitude. That is, negative ties will be rounded towards zero, which differs from the behavior in VCL. Take care when porting code using this rounding mode to VCL.

- **Round to nearest** with other tie-breaking schemes

  There are several other less common arrangements for tie-breaking. These include *ties to odd* (in a similar manner as *ties to even*), random tie-breaking, and stochastic tie-breaking.

  These schemes are not provided in Fastly VCL.

Floating point numbers have more computational nuances than are described by the cursory discussion of biases here. For more details, see What every computer scientist should know about floating-point arithmetic.