# VCL (/vcl)

## Content negotiation (/vcl/content-negotiation/)

### Content negotiation Functions

### accept.charset_lookup() (/vcl/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 (https://httpwg.org/specs/rfc7231.html#rfc.section.5.3.3).

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 (/vcl/types/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() (/vcl/functions/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 (https://httpwg.org/specs/rfc7231.html#rfc.section.5.3.3).

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 (/vcl/types/string/)
accept.encoding_lookup(STRING requested_content_encodings, STRING default, STRING accep
t_header)
```

### Examples

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

## accept.language_filter_basic() (/vcl/functions/accept-language-filter-basic/)

Similar to `accept.language_lookup()` (/vcl/functions/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 (https://tools.ietf.org/html/rfc4647), 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 (/vcl/types/string/)
accept.language_filter_basic(STRING requested_languages, STRING default, STRING accept_
header, INTEGER nmatches)
```

### Examples

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

## accept.language_lookup() (/vcl/functions/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 (https://tools.ietf.org/html/rfc4647), 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 (https://tools.ietf.org/html/rfc4647).

### Format

```
STRING (/vcl/types/string/)
accept.language_lookup(STRING requested_languages, STRING default, STRING accept_heade
r)
```

### Examples

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

## accept.media_lookup() (/vcl/functions/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 (https://httpwg.org/specs/rfc7231.html#rfc.section.5.3.2).

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 (/vcl/types/string/)
accept.media_lookup(STRING requested_media_types, STRING default, STRING range_default
s, 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 (/vcl/cryptographic/)

# Notes

In Base64 decoding, the output theoretically could be in binary but is interpreted as a string. So if the binary output contains '\0' then it could be truncated.

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

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

as a 64bit unsigned integer (little endian) and Base64 encodes the result.

# Examples

## One-Time Password Validation (Token Authentication)

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

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

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

### Example VCL

```
sub vcl_recv {

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

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

#FASTLY recv

  ...
}
```

## Signature

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

## Base64 decoding

A snippet like this in `vcl_error` would set the response body to the value of the request header field named `x-parrot` after Base64-decoding the value:

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

However, if the Base64-decoded string contains a NUL byte (0x00), then that byte and any bytes following it will not be included in the response. Keep that in mind if you intend to send a synthetic response that contains binary data. There is currently no way to send a synthetic response containing a NUL byte.

# Cryptographic Functions

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

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

### Format

```
STRING (/vcl/types/string/)
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() (/vcl/functions/digest-base64-decode/)

Returns the Base64 decoding of the input string, as specified by RFC 4648 (https://tools.ietf.org/html/rfc4648).

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-base64/)

Returns the Base64 encoding of the input string, as specified by RFC 4648 (https://tools.ietf.org/html/rfc4648).

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-base64url-decode/)

Returns the Base64 decoding with URL and filename safe alphabet decoding of the input string, as specified by RFC 4648 (https://tools.ietf.org/html/rfc4648).

## Format

```
STRING (/vcl/types/string/)
digest.base64url_decode(STRING input)
```

## Examples

```
declare local var.base64url_decoded STRING;
set var.base64url_decoded = digest.base64url_decode("zprOsc67z47PgiDOv8-Bzq_Pg86xz4TO
tQ==");
# var.base64url_decoded is now "Καλώς ορίσατε"
```

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

Returns the Base64 decoding with URL and filename safe alphabet decoding of the input string, as specified by RFC 4648 (https://tools.ietf.org/html/rfc4648), without padding (`=`).

## Format

```
STRING (/vcl/types/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-B
zq_Pg86xz4TOtQ");
# var.base64url_nopad_decoded is now "Καλώς ορίσατε"
```

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

Returns the Base64 encoding with URL and filename safe alphabet encoding of the input string, as specified by RFC 4648 (https://tools.ietf.org/html/rfc4648), without padding (`=`).

## Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-base64url/)

Returns the Base64 encoding with URL and filename safe alphabet of the input string, as specified by RFC 4648 (https://tools.ietf.org/html/rfc4648).

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hash-crc32/)

Calculates the 32-bit Cyclic Redundancy Checksum with reversed bit ordering of a string, like that used by bzip2 (https://en.wikipedia.org/wiki/Bzip2). Returns a hex-encoded string in byte-reversed order, e.g. `181989fc` instead of `fc891918`.

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hash-crc32b/)

Calculates the 32-bit Cyclic Redundancy Checksum of a string, as specified by ISO/IEC 13239:2002 (https://www.iso.org/standard/37010.html) and section 8.1.1.6.2 of ITU-T recommendation V.42 (https://www.itu.int/rec/T-REC-V.42-200203-I/en) 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 (/vcl/types/string/)
digest.hash_crc32b(STRING input)
```

### Examples

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

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

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

### Format

```
STRING (/vcl/types/string/)
digest.hash_md5(STRING input)
```

### Examples

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

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

Use the SHA-1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

### Format

```
STRING (/vcl/types/string/)
digest.hash_sha1(STRING input)
```

### Examples

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

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

Use the SHA-224 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hash-sha256/)

Use the SHA-256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hash-sha384/)

Use the SHA-384 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

### Format

```
STRING (/vcl/types/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 "eb455d56d2c1a69de64e832011f3393d45f3fa31d6842f21af92d2fe469c499da5e3179847334a18479c8d1dedea1be3"
```

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

Use the SHA-512 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm) hash. Returns a hex-encoded string.

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hmac-md5-base64/)

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

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hmac-md5/)

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

### Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hmac-sha1-base64/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA-1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a Base64-encoded (https://en.wikipedia.org/wiki/Base64) string.

### Format

```
STRING (/vcl/types/string/)
digest.hmac_sha1_base64(STRING key, STRING input)
```

### Examples

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

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

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA-1 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a hex-encoded string prepended with 0x.

### Format

```
STRING (/vcl/types/string/)
digest.hmac_sha1(STRING key, STRING input)
```

### Examples

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

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

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA-256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a Base64-encoded (https://en.wikipedia.org/wiki/Base64) string.

### Format

```
STRING (/vcl/types/string/)
digest.hmac_sha256_base64(STRING key, STRING input)
```

### Examples

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

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

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA-256 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a hex-encoded string prepended with 0x.

## Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hmac-sha512-base64/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA-512 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a Base64-encoded (https://en.wikipedia.org/wiki/Base64) string.

## Format

```
STRING (/vcl/types/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() (/vcl/functions/digest-hmac-sha512/)

Hash-based message authentication code (https://en.wikipedia.org/wiki/Hash-based_message_authentication_code) using SHA-512 (https://en.wikipedia.org/wiki/Secure_Hash_Algorithm). Returns a hex-encoded string prepended with 0x.

## Format

```
STRING (/vcl/types/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 "0x03ad77c817f32669cccf38dac915d4e55a16833b1ca685979a9df521a0523597809
0043c3da402dad365ed39ac05af4864a804451861f4b4df7680834c6d4f95"
```

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

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

### Format

```
BOOL (/vcl/types/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() (/vcl/functions/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 (/vcl/types/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(user2hashedpass, req.http.User)))) {
  error 401 "Unauthorized";
}
```

## digest.time_hmac_md5() (/vcl/functions/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 (/vcl/types/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() (/vcl/functions/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 and must be non-negative. The `offset` parameter provides a means for mitigating clock skew.

### Format

```
STRING (/vcl/types/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() (/vcl/functions/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 (/vcl/types/string/)
digest.time_hmac_sha256(STRING key, INTEGER interval, INTEGER offset)
```

### Examples

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

## digest.time_hmac_sha512() (/vcl/functions/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 (/vcl/types/string/)
digest.time_hmac_sha512(STRING key, INTEGER interval, INTEGER offset)
```

### Examples

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

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

## Date and time Functions

## parse_time_delta() (/vcl/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 (/vcl/types/integer/)
parse_time_delta(STRING specifier)
```

### Examples

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

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

Converts an integer, representing seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time), 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()` (/vcl/functions/std-time/) instead.

## Format

```
TIME (/vcl/types/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() (/vcl/functions/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 (https://tools.ietf.org/html/rfc822) and RFC 1123 (https://tools.ietf.org/html/rfc1123)
- `Monday, 02-Jan-06 22:04:05 GMT`, RFC 850 (https://tools.ietf.org/html/rfc850)
- `Mon Jan  2 22:04:05 2006`, ANSI-C asctime() (https://www.unix.com/man-page/FreeBSD/3/asctime/)
- `2006-01-02 22:04:05`, an ISO 8601 (https://en.wikipedia.org/wiki/ISO_8601) subset
- `1136239445.00`, seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time)
- `1136239445`, seconds since the UNIX Epoch (https://en.wikipedia.org/wiki/Unix_time)

The only time zone supported is `GMT`.

If the string does not match one of those formats, then the fallback variable is returned instead.

## Format

```
TIME (/vcl/types/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() (/vcl/functions/strftime/)

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

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

### Format

```
STRING (/vcl/types/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() (/vcl/functions/time-add/)

Adds a relative time to a time.

### Format

```
TIME (/vcl/types/time/)
time.add(TIME t1, TIME t2)
```

### Examples

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

## time.hex_to_time() (/vcl/functions/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 (https://en.wikipedia.org/wiki/Unix_time).

### Format

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

### Examples

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

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

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

### Format

```
BOOL (/vcl/types/bool/)
time.is_after(TIME t1, TIME t2)
```

### Examples

```
1 if (time.is_after(time.add(now, 10m), now)) {
2   ...
3 }
```

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

Subtracts a relative time from a time.

### Format

```
TIME (/vcl/types/time/)
time.sub(TIME t1, TIME t2)
```

### Examples

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

## Date and time Variables

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

## now (/vcl/variables/now/)

The current time in RFC 1123 format (https://tools.ietf.org/html/rfc1123) (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`).

### Type

`TIME (/vcl/types/time/)`

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

The time since the request start in milliseconds.

### Type

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

### Accessibility

### Readable From

All subroutines

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

The time since the request start in seconds.

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

The time since the request start in microseconds.

### Type

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

### Accessibility

### Readable From

All subroutines

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

The time since the request started. Also useful for strftime.

### Type

`RTIME (/vcl/types/rtime/)`

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

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

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

### Type

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

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

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

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

### Type

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

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

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

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

### Type

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

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

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

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

### Type

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

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

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

The time the request ended, using RFC 1123 format (https://tools.ietf.org/html/rfc1123) (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`). Also useful for strftime (/vcl/functions/strftime/).

### Type

`TIME (/vcl/types/time/)`

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

The time the request started, after TLS termination, using RFC 1123 format (https://tools.ietf.org/html/rfc1123) (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`).

### Type

`TIME (/vcl/types/time/)`

### Accessibility

### Readable From

All subroutines

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

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

### Type

`RTIME (/vcl/types/rtime/)`

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

# Edge Side Includes (ESI) (/vcl/esi/)

## Edge Side Includes (ESI) Variables

## req.esi (/vcl/variables/req-esi/)

Whether or not to enable ESI processing during this request. Using `set req.esi = true;` will enable ESI processing.

### Type

`BOOL (/vcl/types/bool/)`

### Accessibility

### Readable From

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

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

# Using geographic variables with shielding

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

```
1  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:

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

In general strings in VCL may be *not set* (see the VCL documentation for types (/vcl/types/string/)). 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 (https://en.wikipedia.org/wiki/Private_network) (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 (https://tools.ietf.org/html/rfc5737) 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 |
| `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 (/vcl/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 (/vcl/types/string/)`

### Accessibility

### Readable From

All subroutines

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

Autonomous system (https://en.wikipedia.org/wiki/Autonomous_system_(Internet)) (AS) number.

The `INTEGER` type in VCL is wide enough (/vcl/types/integer/) 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 (https://tools.ietf.org/html/rfc5396) 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;
```

For example, the 32-bit AS number 65550 (reserved by RFC 5398 (https://tools.ietf.org/html/rfc5398) 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 (/vcl/types/integer/)

### Accessibility

### Readable From

All subroutines

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

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

### Type

INTEGER (/vcl/types/integer/)

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

## client.geo.city.latin1 (/vcl/variables/client-geo-city-latin1/)

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

## client.geo.conn_speed (/vcl/variables/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 (https://en.wikipedia.org/wiki/Optical_Carrier_transmission_rates) and T-carrier (https://en.wikipedia.org/wiki/T-carrier) for background on OC- and T- connections.

### Type

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

### Accessibility

### Readable From

All subroutines

## client.geo.continent_code (/vcl/variables/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 (https://unstats.un.org/unsd/methodology/m49/).

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 (/vcl/types/string/)`

### Accessibility

### Readable From

All subroutines

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

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

These values include subdivisions that are assigned their own country codes in ISO 3166-1 (https://en.wikipedia.org/wiki/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.

The following VCL fragment uses a two-letter country code to construct an emoji flag from its corresponding Unicode regional indicator symbols (https://en.wikipedia.org/wiki/Regional_Indicator_Symbol):

```
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 (/vcl/types/string/)`

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

Country name, encoded using ASCII encoding.

This field is a lowercase transliteration of the ISO 3166-1 (https://en.wikipedia.org/wiki/ISO_3166-1) English short name for a country.

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 (/vcl/types/string/)`

### Accessibility

### Readable From

All subroutines

# client.geo.country_name.latin1 (/vcl/variables/client-geo-country-name-latin1/)

Country name, encoded using Latin-1 encoding.

### Type

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

### Accessibility

### Readable From

All subroutines

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

Country name, encoded using UTF-8 encoding.

This field is the ISO 3166-1 (https://en.wikipedia.org/wiki/ISO_3166-1) English short name for a country.

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

# client.geo.gmt_offset (/vcl/variables/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 (https://en.wikipedia.org/wiki/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.

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, "..$", "")); # tru
ncate
```

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 (/vcl/types/integer/)`

### Accessibility

### Readable From

All subroutines

## client.geo.ip_override (/vcl/variables/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 (/vcl/types/ip/)`

### Accessibility

### Readable From

All subroutines

## client.geo.latitude (/vcl/variables/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 (https://en.wikipedia.org/wiki/World_Geodetic_System) coordinate reference system.

An example showing construction of a geo URI (https://en.wikipedia.org/wiki/Geo_URI_scheme) as specified by RFC 5870 (https://tools.ietf.org/html/rfc5870) 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 (https://en.wikipedia.org/wiki/Geographical_zone) 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 (https://en.wikipedia.org/wiki/Prime_(symbol)), not quotes.

### Type

`FLOAT (/vcl/types/float/)`

### Accessibility

### Readable From

All subroutines

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

Longitude, in units of degrees from the IERS Reference Meridian (https://en.wikipedia.org/wiki/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 (https://en.wikipedia.org/wiki/World_Geodetic_System) coordinate reference system.

### Type

`FLOAT (/vcl/types/float/)`

### Accessibility

### Readable From

All subroutines

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

Metro code.

Metro codes represent designated market areas (https://www.nielsen.com/intl-campaigns/us/dma-maps.html) (DMAs) in the United States.

### Type

`INTEGER (/vcl/types/integer/)`

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

ISO 3166-2 (https://en.wikipedia.org/wiki/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 (https://en.wikipedia.org/wiki/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;
  }
}
```

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 (/vcl/types/string/)`

### Accessibility

### Readable From

All subroutines

## client.geo.region.latin1 (/vcl/variables/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` (/vcl/variables/client-geo-region-ascii/).

### Type

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

### Accessibility

### Readable From

All subroutines

## client.geo.region.utf8 (/vcl/variables/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` (/vcl/variables/client-geo-region-ascii/).

### Type

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

### Accessibility

### Readable From

All subroutines

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

Alias of `client.geo.region.ascii` (/vcl/variables/client-geo-region-ascii/).

### Type

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

### Accessibility

### Readable From

All subroutines

# Miscellaneous (/vcl/miscellaneous/)

# Miscellaneous features

| Feature | Description |
| --- | --- |
| `goto` | Performs a one-way transfer of control to another line of code. See the example for more information. |

| Feature | Description |
| --- | --- |
| `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.

```
1  sub vcl_recv {
2    if (!req.http.Foo) {
3      goto foo;
4    }
5
6  foo:
7    set req.http.Foo = "1";
8  }
```

## Return

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

```
1  sub custom_subroutine {
2    if (req.http.Cookie:user_id) {
3      return;
4    }
5
6    # do a bunch of other stuff
7  }
```

## Miscellaneous Functions

## addr.extract_bits() (/vcl/functions/addr-extract-bits/)

Extract `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 (/vcl/types/integer/)
addr.extract_bits(IP, start_bit INTEGER, bit_count INTEGER)
```

### Examples

```
1 if (addr.extract_bits(server.ip, 0, 8) == 7) {
2   # received on an IPv4 address that ends in ".7" or an IPv6 address that ends in "0
3 7"
  }
```

## addr.is_ipv4() (/vcl/functions/addr-is-ipv4/)

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

### Format

```
BOOL (/vcl/types/bool/)
addr.is_ipv4(IP ip)
```

### Examples

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

## addr.is_ipv6() (/vcl/functions/addr-is-ipv6/)

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

### Format

```
BOOL (/vcl/types/bool/)
addr.is_ipv6(IP ip)
```

### Examples

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

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

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

> ★ **TIP:** If you are escaping JSON strings, use `json.escape()` (/vcl/functions/json-escape/) instead.

### Format

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

### Examples

```
1  declare local var.escaped STRING;
2  set var.escaped = "city=%22" + cstr_escape(client.geo.city.ascii) + "%22";
3  # var.escaped is now e.g. city="london"
```

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

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

### Format

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

### Examples

```
1  if (http_status_matches(beresp.status, "!200,304")) {
2    restart;
3  }
```

# if() (/vcl/functions/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 (/vcl/types/string/)
if(BOOL expression, STRING value-when-true, STRING value-when-false)
```

### Examples

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

# json.escape() (/vcl/functions/json-escape/)

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

### Format

```
STRING (/vcl/types/string/)
json.escape(STRING string)
```

### Examples

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

# regsub() (/vcl/functions/regsub/)

Replaces the first occurrence of `pattern`, which may be 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 (/vcl/types/string/)
regsub(STRING input, STRING pattern, STRING replacement)
```

### Examples

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

## regsuball() (/vcl/functions/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("aa", "a", "aa")` will return a string "aaaa" 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 (/vcl/types/string/)
regsuball(STRING input, STRING pattern, STRING replacement)
```

### Examples

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

## setcookie.get_value_by_name() (/vcl/functions/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 (https://tools.ietf.org/html/rfc6265#section-4.1.1).

### Format

```
STRING (/vcl/types/string/)
setcookie.get_value_by_name(ID where, STRING cookie_name)
```

### Examples

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

## std.anystr2ip() (/vcl/functions/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()` (/vcl/functions/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`.

### Format

```
IP (/vcl/types/ip/)
std.anystr2ip(STRING addr, STRING fallback)
```

### Examples

```
1  if (std.anystr2ip("0xc0.0.01001", "192.0.2.2") ~ my_acl) {
2    ...
3  }
```

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

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

### Format

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

### Examples

```
1  if (std.atoi(req.http.X-Decimal) == 42) {
2    set req.http.X-TheAnswer = "Found";
3  }
```

## std.collect() (/vcl/functions/std-collect/)

Combine 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 (/vcl/types/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"
```

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

An alias of `std.str2ip()` (/vcl/functions/std-str2ip/).

### Format

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

### Examples

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

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

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

### Format

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

### Examples

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

## std.prefixof() (/vcl/functions/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 (/vcl/types/bool/)
std.prefixof(STRING s, STRING begins_with)
```

### Examples

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

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

Converts the string representation of an IP address (IPv4 or IPv6) into an `IP type` (/vcl/types/ip/). 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.

### Format

```
IP (/vcl/types/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() (/vcl/functions/std-strlen/)

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

### Format

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

### Examples

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

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

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

### Format

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

### Examples

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

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

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

### Format

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

### Examples

```
1  if (std.strtol(req.http.X-HexValue, 16) == 42) {
2    set req.http.X-TheAnswer = "Found";
3  }
```

# std.suffixof() (/vcl/functions/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 (/vcl/types/bool/)
std.suffixof(STRING s, STRING ends_with)
```

### Examples

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

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

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

### Format

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

### Examples

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

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

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

### Format

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

### Examples

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

## subfield() (/vcl/functions/subfield/)

Provides a means to access subfields from a header like `Cache-Control`, `Cookie`, and `Edge-Control` 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 (/vcl/types/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", "&");
```

## urldecode() (/vcl/functions/urldecode/)

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

### Format

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

### Examples

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

## urlencode() (/vcl/functions/urlencode/)

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

### Format

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

### Examples

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

# Miscellaneous Variables

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

Same as `req.url.basename` (/vcl/variables/req-url-basename/), except for use between Fastly and your origin servers.

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

`IP (/vcl/types/ip/)`

### Accessibility

### Readable From

- `vcl_fetch`

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

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

### Type

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

### Accessibility

### Readable From

- `vcl_fetch`

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

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

### Type

`INTEGER (/vcl/types/integer/)`

### Accessibility

### Readable From

- `vcl_fetch`

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

Defines how long an object can remain overdue and still have Varnish consider it for grace mode. Fastly has implemented `stale-if-error` (/guides/performance-tuning/serving-stale-content#manually-enabling-serve-stale) as a parallel implementation of `beresp.grace`.

### Type

`RTIME (/vcl/types/rtime/)`

### Accessibility

### Readable From

- `vcl_fetch`

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

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

### Type

`BOOL (/vcl/types/bool/)`

### Accessibility

### Readable From

- `vcl_fetch`

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

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

### Type

`BOOL (/vcl/types/bool/)`

### Accessibility

### Readable From

- `vcl_fetch`

## client.ip (/vcl/variables/client-ip/)

The IP address of the client making the request.

### Type

`IP (/vcl/types/ip/)`

### Accessibility

### Readable From

All subroutines

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

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

### Type

`INTEGER (/vcl/types/integer/)`

### Accessibility

### Readable From

All subroutines

## client.requests (/vcl/variables/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 (/vcl/types/integer/)`

### Accessibility

### Readable From

All subroutines

## client.socket.pace (/vcl/variables/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 (/vcl/types/integer/)`

### Accessibility

### Readable From

All subroutines

## req.body.base64 (/vcl/variables/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 (/vcl/types/string/)`

### Accessibility

### Readable From

All subroutines

## req.body (/vcl/variables/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 (/vcl/types/string/)`

### Accessibility

### Readable From

All subroutines

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

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

### Type

`RTIME (/vcl/types/rtime/)`

### Accessibility

### Readable From

All subroutines

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

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

Indicates whether the request was made using IPv6 or not.

### Type

`BOOL (/vcl/types/bool/)`

### Accessibility

### Readable From

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

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

Counts the number of times the VCL has been restarted.

### Type

`INTEGER (/vcl/types/integer/)`

### Accessibility

### Readable From

All subroutines

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

The file name specified in a URL.

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

The directories specified in a URL.

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

The file extension specified in a URL.

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

The full path, without any query parameters.

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

The full path, including query parameters.

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

# Examples

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

```
1  sub vcl_recv {
2      # return this URL with only the parameters that match this regular expression
3      set req.url = querystring.regfilter_except(req.url, "^(q|p)$");
4  }
```

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:

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

## Query string manipulation Functions

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

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

### Format

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

### Examples

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

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

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

### Format

```
STRING (/vcl/types/string/)
querystring.add(STRING, STRING, STRING)
```

### Examples

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

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

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

### Format

```
STRING (/vcl/types/string/)
querystring.clean(STRING)
```

### Examples

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

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

Returns the given URL but only keeps the listed parameters.

### Format

```
STRING (/vcl/types/string/)
querystring.filter_except(STRING, STRING_LIST)
```

### Examples

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

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

Returns the given URL without the listed parameters.

### Format

```
STRING (/vcl/types/string/)
querystring.filter(STRING, STRING_LIST)
```

### Examples

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

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

Returns the separator needed by the `querystring.filter()` (/vcl/functions/querystring-filter/) and `querystring.filter_except()` (/vcl/functions/querystring-filter-except/) functions.

### Format

```
STRING (/vcl/types/string/)
querystring.filtersep()
```

### Examples

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

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

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

### Format

```
STRING (/vcl/types/string/)
querystring.globfilter_except(STRING, STRING)
```

### Examples

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

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

Returns the given URL without the parameters matching a glob.

### Format

```
STRING (/vcl/types/string/)
querystring.globfilter(STRING, STRING)
```

### Examples

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

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

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

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

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 (/vcl/types/string/)
querystring.regfilter_except(STRING, STRING)
```

### Examples

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

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

Returns the given URL without the parameters matching a regular expression. 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.gr
2  oup.2
3    set req.url = querystring.regfilter(req.url, "key-([0-9]|\w)"); # does not captur
4  e
5    set req.http.X-Key-1 = re.group.1;
     set req.http.X-Key-2 = re.group.2;
   }
```

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 (/vcl/types/string/)
querystring.regfilter(STRING, STRING)
```

### Examples

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

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

Returns the given URL with its query-string removed.

### Format

```
STRING (/vcl/types/string/)
querystring.remove(STRING)
```

### Examples

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

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

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

### Format

```
STRING (/vcl/types/string/)
querystring.set(STRING, STRING, STRING)
```

### Examples

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

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

Returns the given URL with its query-string sorted.

### Format

```
STRING (/vcl/types/string/)
querystring.sort(STRING)
```

### Examples

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

# Randomness (/vcl/randomness/)

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

# Random strings

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

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

# Random content cookies in pure VCL

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

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

# Random decisions

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

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

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

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

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

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

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

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

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

## Randomness Functions

### randombool_seeded() (/vcl/functions/randombool-seeded/)

Identical to randombool (/vcl/functions/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 (/vcl/types/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() (/vcl/functions/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 (/vcl/types/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() (/vcl/functions/randomint-seeded/)

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

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

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

### Format

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

### Examples

```
 1  if (randomint_seeded(1, 5, user_id) < 5) {
 2    set req.http.X-ABTest = "A";
 3  } else {
 4    set req.http.X-ABTest = "B";
 5  }
 6  if (randomint_seeded(-1, 0, 555) == -1) {
 7    set req.http.X-ABTest = "A";
 8  } else {
 9    set req.http.X-ABTest = "B";
10  }
```

## randomint() (/vcl/functions/randomint/)

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

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

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

### Format

```
INTEGER (/vcl/types/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() (/vcl/functions/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 (/vcl/types/string/)
randomstr(INTEGER len, STRING characters)
```

### Examples

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

# Server (/vcl/server/)

## Server Variables

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

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

### Type

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

### Accessibility

### Readable From

All subroutines

# server.hostname (/vcl/variables/server-hostname/)

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

### Type

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

### Accessibility

### Readable From

All subroutines

# server.identity (/vcl/variables/server-identity/)

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

### Type

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

### Accessibility

### Readable From

All subroutines

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

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

| Region Name | Approximate Geographic Location of Fastly POPs |
|---|---|
| APAC | Australia and New Zealand |
| Asia | throughout the Asian continent (except India) |
| Asia-South | southern Asia |
| EU-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-South | southern South America |
| South-Africa | the southern regions of Africa |

| Region Name | Approximate Geographic Location of Fastly POPs |
|---|---|
| US-Central | the central United States |
| US-East | the eastern United States |
| US-West | the western United States |

### Type

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

### Accessibility

### Readable From

All subroutines

# Size (/vcl/size/)

## Size Variables

## beresp.body_bytes_written (/vcl/variables/beresp-body-bytes-written/)

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

### Type

`INTEGER (/vcl/types/integer/)`

### Accessibility

### Readable From

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

## beresp.header_bytes_written (/vcl/variables/beresp-header-bytes-written/)

Total header bytes written to a backend.

### Type

`INTEGER (/vcl/types/integer/)`

### Accessibility

### Readable From

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

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

Total body bytes read from the client generating the request.

### Type

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

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

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

Total bytes read from the client generating the request.

### Type

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

### Accessibility

### Readable From

- `vcl_deliver`
- `vcl_log`

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

Total header bytes read from the client generating the request.

### Type

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

### Accessibility

### Readable From

All subroutines

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

Body bytes to send to the client in the response.

### Type

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

### Accessibility

### Readable From

- `vcl_log`

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

Total bytes to send to the client in the response.

### Type

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

### Accessibility

### Readable From

- `vcl_log`

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

Whether the response completed successfully or not.

### Type

`BOOL (/vcl/types/bool/)`

### Accessibility

### Readable From

- `vcl_log`

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

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

### Type

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

### Accessibility

### Readable From

- `vcl_log`

# Table (/vcl/table/)

Tables are declared as follows:

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

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() (/vcl/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 (/vcl/types/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 (/vcl/tls-and-http2/)

When using these variables, remember the following:

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

## TLS and HTTP/2 Functions

## h2.disable_header_compression() (/vcl/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 (/vcl/types/void/)
h2.disable_header_compression(STRING header)
```

### Examples

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

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

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

### Format

```
VOID (/vcl/types/void/)
h2.push(STRING resource)
```

### Examples

```
1  if (fastly_info.is_h2 && req.url == "/") {
2    h2.push("/assets/jquery.js");
3  }
```

## TLS and HTTP/2 Variables

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

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

### Type

`BOOL (/vcl/types/bool/)`

### Accessibility

### Readable From

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

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

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

### Type

`INTEGER (/vcl/types/integer/)`

### Accessibility

### Readable From

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

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

Whether or not the request was made using http2.

### Type

`BOOL (/vcl/types/bool/)`

### Accessibility

### Readable From

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

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

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

### Type

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

### Accessibility

### Readable From

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

## tls.client.ciphers_list_sha (/vcl/variables/tls-client-ciphers-list-sha/)

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

### Type

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

### Accessibility

### Readable From

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

## tls.client.ciphers_list_txt (/vcl/variables/tls-client-ciphers-list-txt/)

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

### Type

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

### Accessibility

### Readable From

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

## tls.client.ciphers_list (/vcl/variables/tls-client-ciphers-list/)

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

### Type

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

### Accessibility

### Readable From

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

## tls.client.ciphers_sha (/vcl/variables/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 (/vcl/types/string/)`

### Accessibility

### Readable From

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

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

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

### Type

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

### Accessibility

### Readable From

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

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

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

### Type

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

### Accessibility

### Readable From

- `vcl_recv`
- `vcl_hash`

- `vcl_deliver`
- `vcl_log`

## tls.client.tlsexts_list_sha (/vcl/variables/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 (/vcl/types/string/)`

### Accessibility

### Readable From

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

## tls.client.tlsexts_list_txt (/vcl/variables/tls-client-tlsexts-list-txt/)

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

### Type

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

### Accessibility

### Readable From

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

## tls.client.tlsexts_list (/vcl/variables/tls-client-tlsexts-list/)

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

### Type

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

### Accessibility

### Readable From

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

## tls.client.tlsexts_sha (/vcl/variables/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 (/vcl/types/string/)`

### Accessibility

### Readable From

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

# UUID (/vcl/uuid/)

## UUID Functions

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

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

### Format

```
STRING (/vcl/types/string/)
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() (/vcl/functions/uuid-is-valid/)

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

### Format

```
BOOL (/vcl/types/bool/)
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() (/vcl/functions/uuid-is-version3/)

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

### Format

```
BOOL (/vcl/types/bool/)
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() (/vcl/functions/uuid-is-version4/)

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

### Format

```
BOOL (/vcl/types/bool/)
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() (/vcl/functions/uuid-is-version5/)

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

### Format

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

### Examples

```
1 if (uuid.is_version5(req.http.X-Unique-Id)) {
2   set beresp.http.X-Unique-Id-Valid-V5 = "yes";
3 }
```

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

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

### Format

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

### Examples

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

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

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

### Format

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

### Examples

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

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

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

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

### Format

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

### Examples

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

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

Returns a UUID based on random number generator output.

### Format

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

### Examples

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

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

Derives a UUID corresponding to `name` within the given `namespace` using 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 (/vcl/types/string/)
uuid.version5(STRING namespace, STRING name)
```

### Examples

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

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

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

### Format

```
STRING (/vcl/types/string/)
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 (/vcl/custom-vcl/creating-custom-vcl/)

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

- **You can only restart Varnish requests three times.** This limit exists to prevent infinite loops.
- **VCL doesn't take kindly to Windows newlines (line breaks).** It's best to avoid them entirely.

- **It's best to use `curl -X PURGE` to initiate purges via API (/api/purge).** To restrict access to purging, check for the `FASTLYPURGE` method not the `PURGE` method. When you send a request to Varnish to initiate a purge, the HTTP method that you use is "PURGE", but it has already been changed to "FASTLYPURGE" by the time your VCL runs that request.
- **If you override TTLs with custom VCL, your default TTL set in the configuration (/guides/performance-tuning/serving-stale-content) will not be honored** and the expected behavior may change.

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

# Inserting custom VCL in Fastly's VCL boilerplate

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

## Custom VCL placement example

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

## Fastly's VCL boilerplate

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

```
sub vcl_recv {
#FASTLY recv

  if (req.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.m
ethod == "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.h
ttp.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
```

```
54
55    if (!obj.cacheable) {
56      return(pass);
57    }
58    return(deliver);
59  }
60
61  sub vcl_miss {
62  #FASTLY miss
63    return(fetch);
64  }
65
66  sub vcl_deliver {
67  #FASTLY deliver
68    return(deliver);
69  }
70
71  sub vcl_error {
72  #FASTLY error
73  }
74
75  sub vcl_pass {
76  #FASTLY pass
77  }
78
    sub vcl_log {
    #FASTLY log
    }
```

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

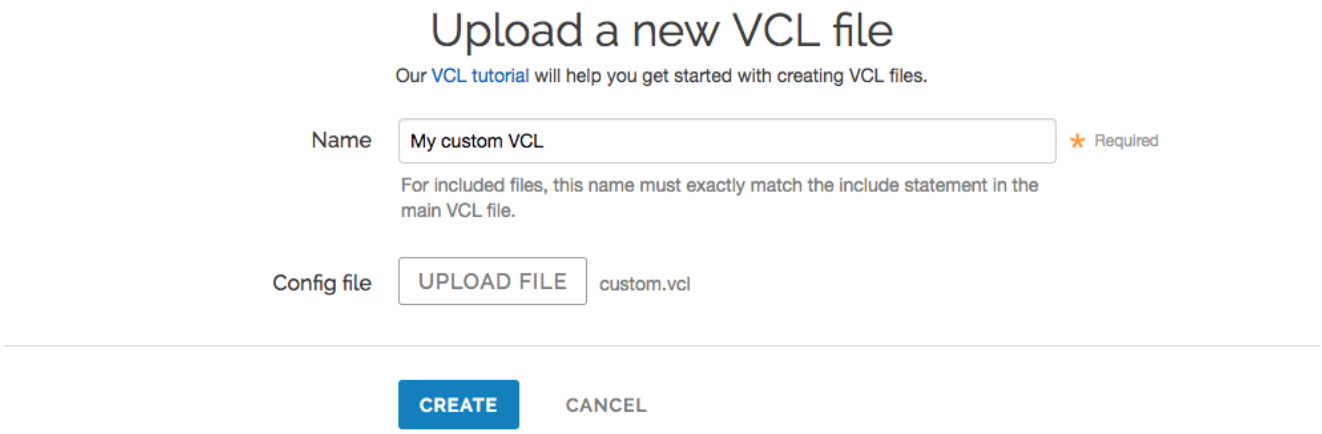
Fastly allows you create your own Varnish Configuration Language (VCL) files with specialized configurations. By uploading custom VCL files, you can use custom VCL and Fastly VCL together at the same time (/vcl/custom-vcl/creating-custom-vcl/). Keep in mind that your custom VCL always takes precedence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

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

## Uploading a VCL file

Follow these instructions to upload a custom VCL file:

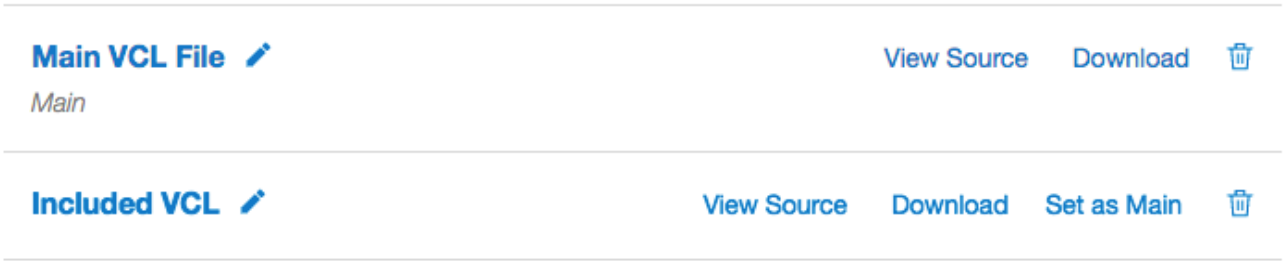
1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **Configuration** button and then select **Clone active**. The Domains page appears.
4. Click the **Custom VCL** tab. The Custom VCL page appears.
5. Click the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




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

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

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

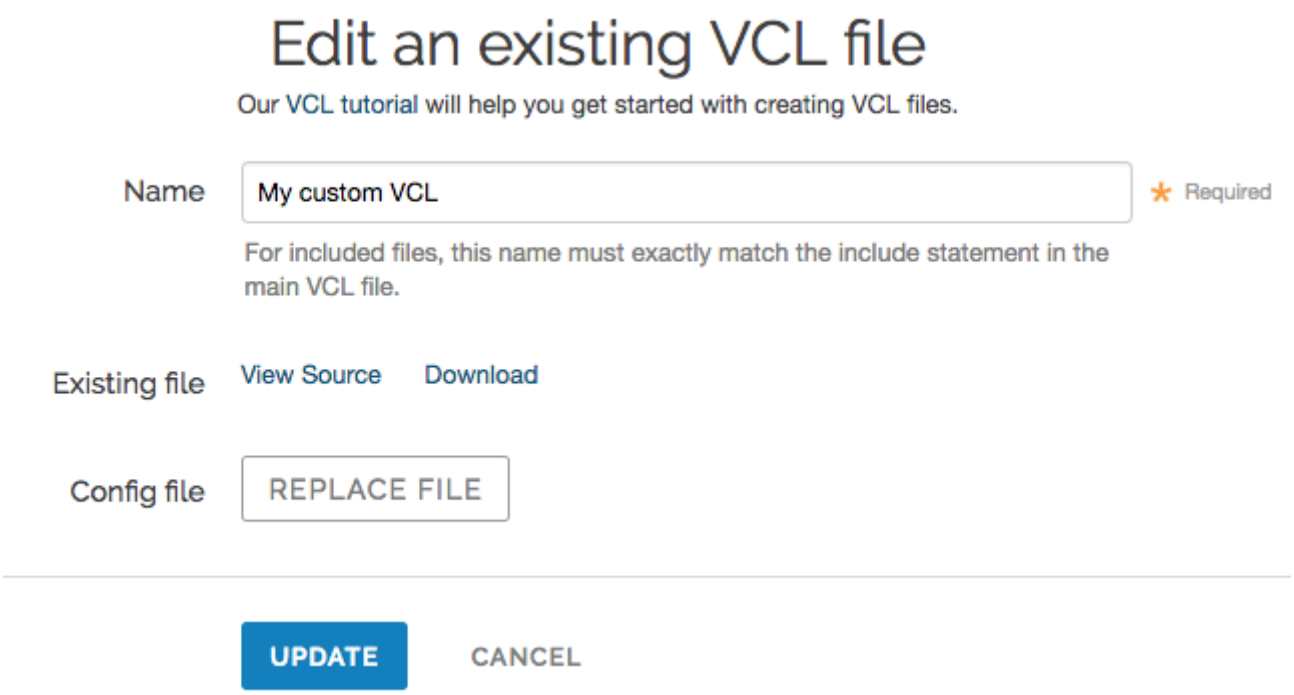



9. Click the **Activate** button to deploy your configuration changes.

# Editing a VCL file

To edit an existing VCL file, follow these instructions:

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




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

# Including additional VCL configurations

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

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

```
include "ACL"
```

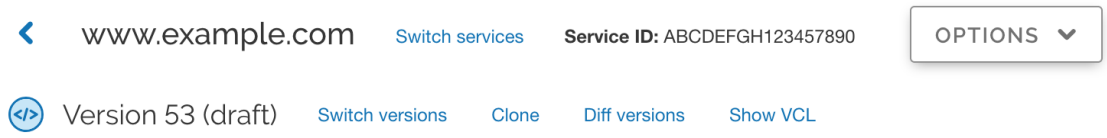
# Previewing and testing VCL (/vcl/custom-vcl/previewing-and-testing-vcl/)

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

## Previewing VCL before activation

To preview VCL prior to activating a service version.

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.
3. Click the **Configuration** button and then select **Clone active**. The Domains page appears.
4. Click the **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 (/vcl/custom-vcl/uploading-custom-vcl/) to the production service you'll be using.

# § VCL Snippets

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

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

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

### Limitations of VCL Snippets

- Snippets are limited to 1MB in size by default. If you need to store snippets larger than the limit, contact support@fastly.com (mailto:support@fastly.com).
- Snippets don’t currently support conditions created through the web interface. You can, however, use `if` statements (/vcl/functions/if/) in snippet code.
- Snippets cannot currently be shared between services.

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

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

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

---

# Creating and using a dynamic VCL Snippet

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

```
curl -X POST -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippet -H "Fastly-Key:FASTLY_API_TOKEN" -H 'Content-Type: application/x-www-form-urlencoded' --data $'name=my_dynamic_snippet_name&type=recv&dynamic=1&content=if ( req.url ) {\n set req.http.my-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.

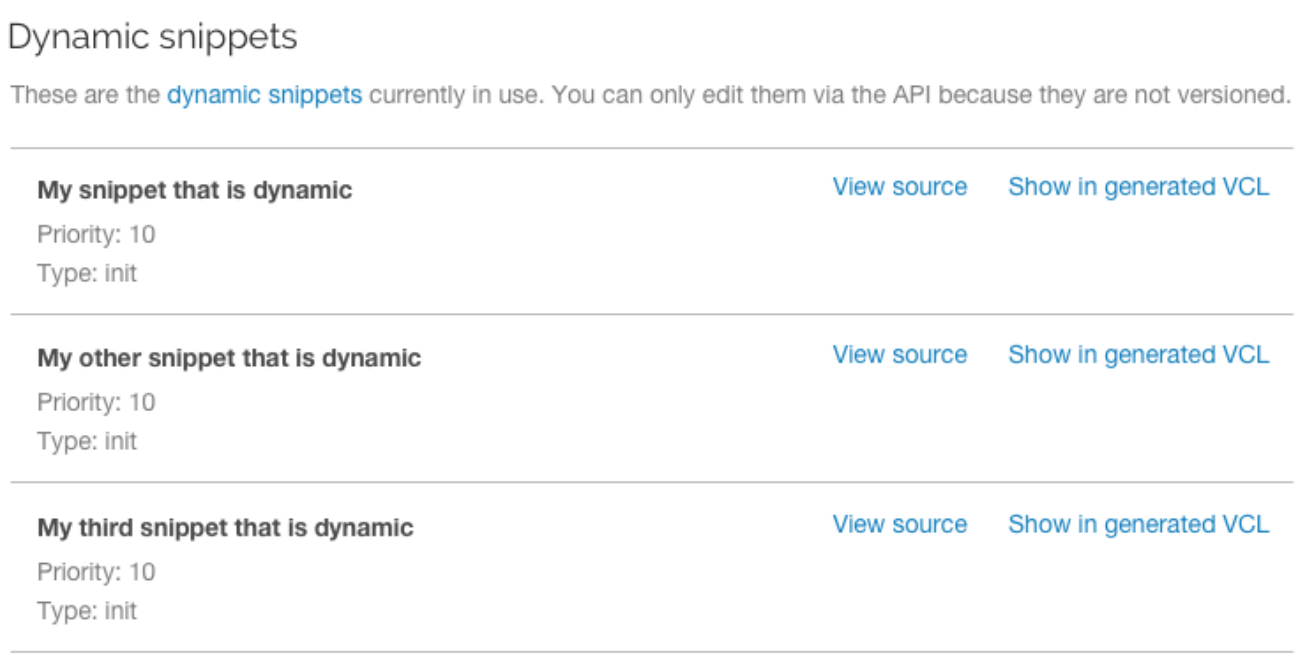
# Viewing dynamic VCL Snippets in the web interface

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

# Updating an existing dynamic VCL Snippet

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

```
curl -X PUT -s https://api.fastly.com/service/<Service ID>/snippet/<my_dynamic_snippet_id> -H "Fastly-Key:FASTLY_API_TOKEN" -H 'Content-Type: application/x-www-form-urlencoded' --data $'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 (/vcl/vcl-snippets/using-regular-vcl-snippets/)

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

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

## Creating a regular VCL Snippet

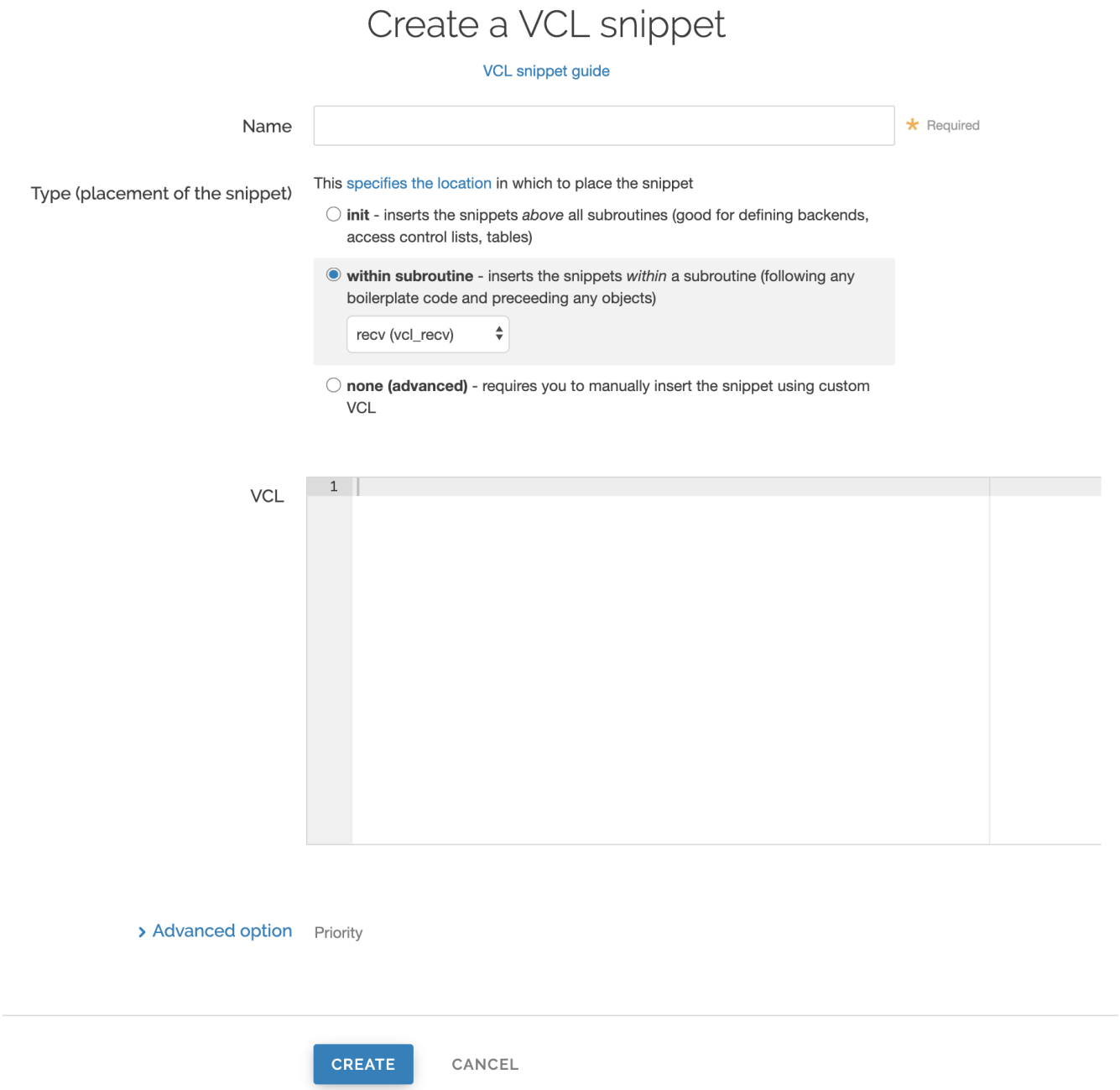
You can create regular VCL Snippets via the web interface or via the API.

### Via the web interface

To create a regular VCL Snippet via the web interface:

1. Log in to the Fastly web interface and click the **Configure** link.
2. From the service menu, select the appropriate service.

3. Click the **VCL Snippets** link. The VCL Snippets page appears.
4. Click **Create Snippet**. The Create a VCL snippet page appears.




5. In the **Name** field, type an appropriate name (for example, `Example Snippet`).
6. Using the **Type** controls, select the location in which the snippet should be placed as follows:
   - Select `init` to insert it above all subroutines in your VCL.
   - Select `within subroutine` to insert it within a specific subroutine and then select the specific subroutine from the **Select subroutine** menu.
   - Select `none (advanced)` to insert it manually. See Including regular snippets in custom VCL (/vcl/vcl-snippets/using-regular-vcl-snippets/#including-regular-snippets-in-custom-

vcl) for the additional manual insertion requirements if you select this option.

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

## Via the API

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

```
curl -X POST -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/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 (/vcl/vcl-snippets/using-dynamic-vcl-snippets/). In addition, the returned JSON includes a populated `content` field because the snippet content is stored in a versioned object.

# Viewing regular VCL Snippets in the web interface

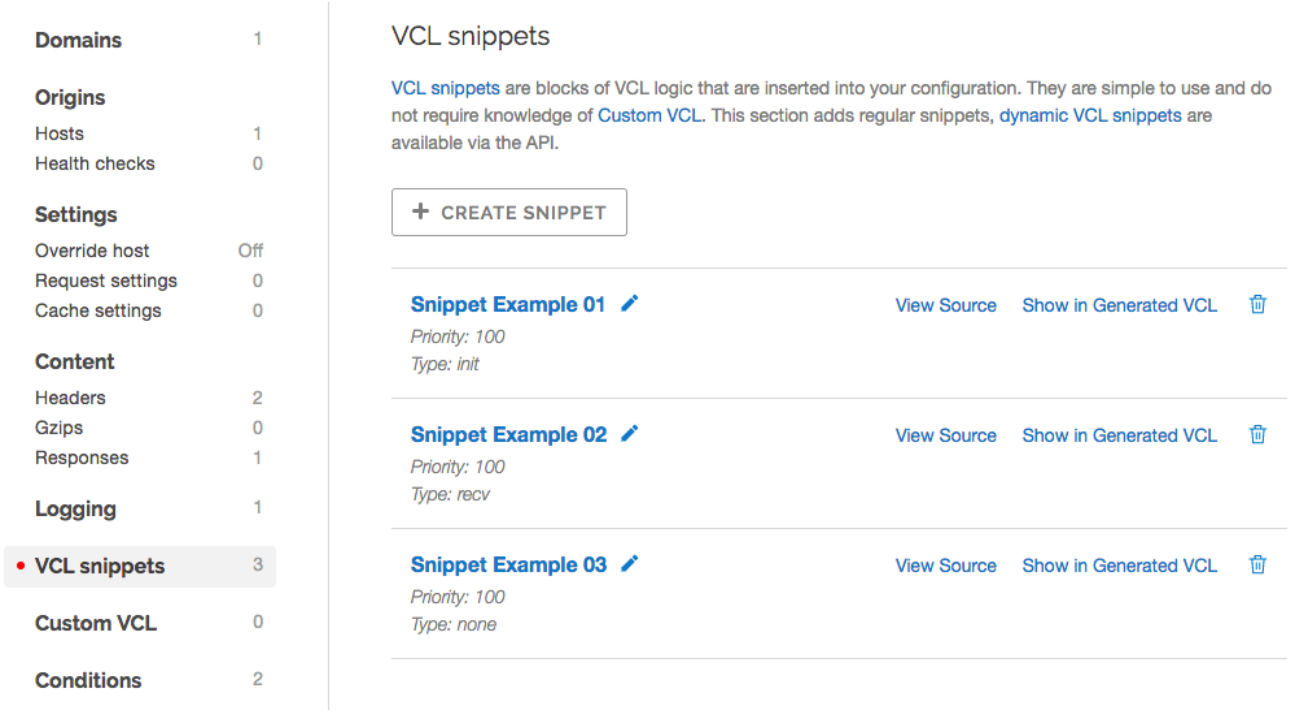
You can view a list of regular VCL snippets. You can also view just the source of a specific snippet or a specific snippet's location in generated VCL.

## Viewing a list of regular VCL Snippets

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

1. Log in to the Fastly web interface and click the **Configure** link.

2. From the service menu, select the appropriate service.
3. Click the **VCL Snippets** link. The VCL Snippets page appears listing all available VCL snippets for your service.




## Viewing the source of a specific snippet

You can view just the source of a specific snippet:

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

## Viewing the location of a specific snippet in generated VCL

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

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

# Fetching regular VCL Snippets via the API

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

## Fetching an individual regular VCL Snippet

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

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

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

## Fetching a list of regular VCL Snippets

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

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

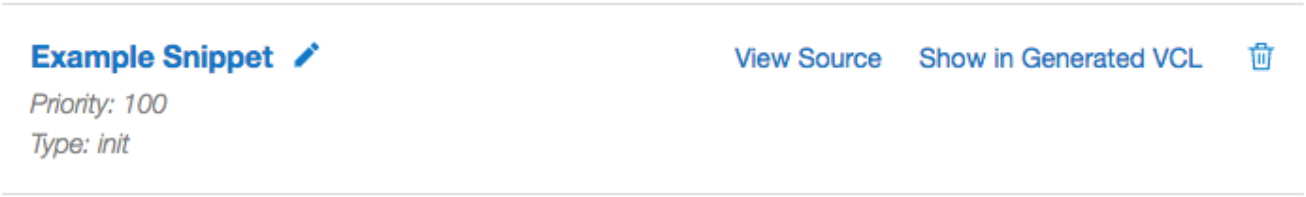
# Updating an existing regular VCL Snippet

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

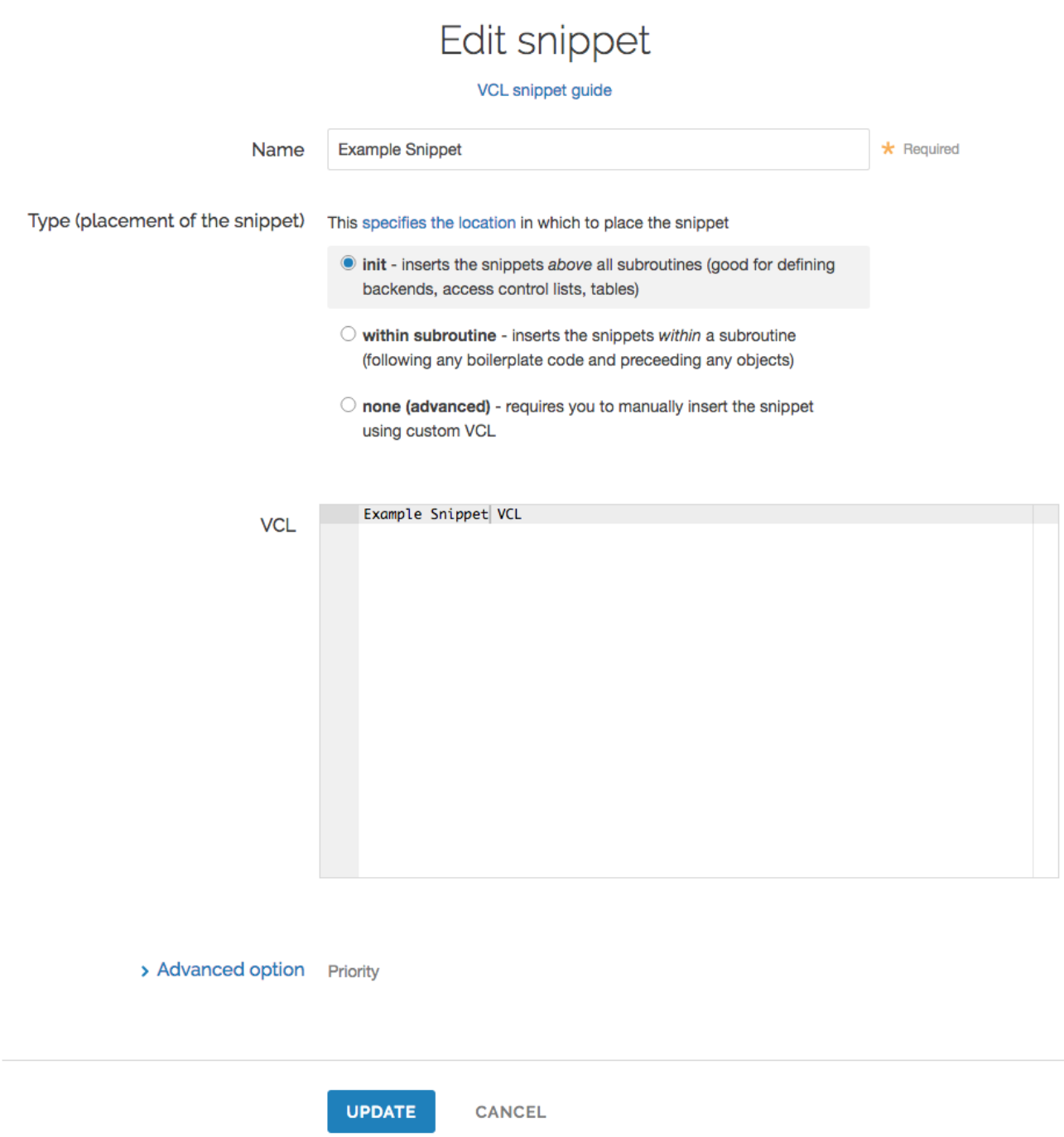
## Via the web interface

To update an individual snippet via the web interface:

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




The Edit snippet page appears.




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

## Via the API

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

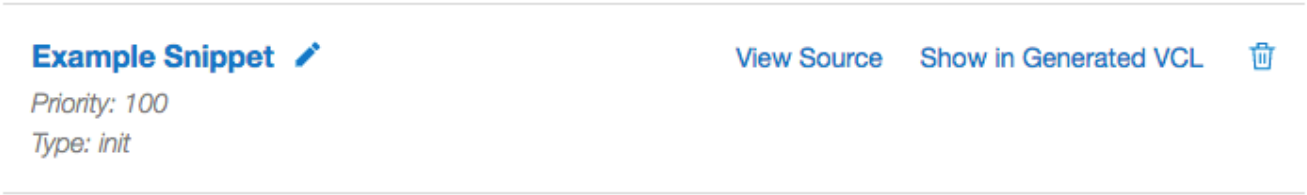
```
curl -X PUT -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippet/<Snippet Name e.g my_regular_snippet> -H "Fastly-Key:FASTLY_API_TOKEN" -H 'Content-Type: application/x-www-form-urlencoded' --data $'content=if ( req.url ) {\n set req.http.my-snippet-test-header = \"affirmative\";\n}';
```

# Deleting an existing regular VCL Snippet

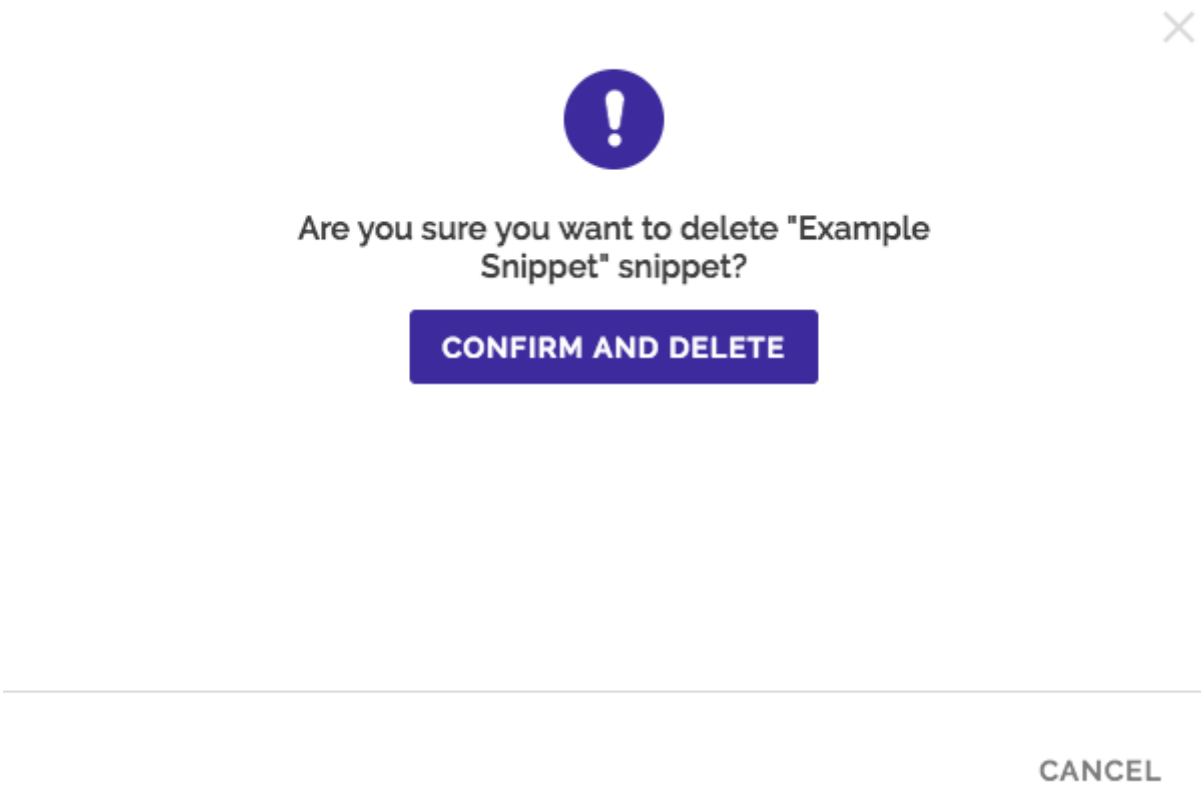
You can update existing regular VCL Snippets via the web interface or via the API.

## 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:

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

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

# § VCL Reference

# Functions (/vcl/functions/)

These VCL functions are supported by Fastly.

## Content negotiation (/vcl/content-negotiation/)

Functions for selecting a response from common content negotiation request headers.

- accept.charset_lookup() (/vcl/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.
- accept.encoding_lookup() (/vcl/functions/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() (/vcl/functions/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() (/vcl/functions/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() (/vcl/functions/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 (/vcl/cryptographic/)

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

- digest.awsv4_hmac() (/vcl/functions/digest-awsv4-hmac/) — Returns an AWSv4 message authentication code based on the supplied `key` and `string`.
- digest.base64_decode() (/vcl/functions/digest-base64-decode/) — Returns the Base64 decoding of the input string, as specified by RFC 4648.
- digest.base64() (/vcl/functions/digest-base64/) — Returns the Base64 encoding of the input string, as specified by RFC 4648.

- digest.base64url_decode() (/vcl/functions/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() (/vcl/functions/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() (/vcl/functions/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() (/vcl/functions/digest-base64url/) — Returns the Base64 encoding with URL and filename safe alphabet of the input string, as specified by RFC 4648.
- digest.hash_crc32() (/vcl/functions/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() (/vcl/functions/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() (/vcl/functions/digest-hash-md5/) — Use the MD5 hash.
- digest.hash_sha1() (/vcl/functions/digest-hash-sha1/) — Use the SHA-1 hash.
- digest.hash_sha224() (/vcl/functions/digest-hash-sha224/) — Use the SHA-224 hash.
- digest.hash_sha256() (/vcl/functions/digest-hash-sha256/) — Use the SHA-256 hash.
- digest.hash_sha384() (/vcl/functions/digest-hash-sha384/) — Use the SHA-384 hash.
- digest.hash_sha512() (/vcl/functions/digest-hash-sha512/) — Use the SHA-512 hash.
- digest.hmac_md5_base64() (/vcl/functions/digest-hmac-md5-base64/) — Hash-based message authentication code using MD5.
- digest.hmac_md5() (/vcl/functions/digest-hmac-md5/) — Hash-based message authentication code using MD5.
- digest.hmac_sha1_base64() (/vcl/functions/digest-hmac-sha1-base64/) — Hash-based message authentication code using SHA-1.
- digest.hmac_sha1() (/vcl/functions/digest-hmac-sha1/) — Hash-based message authentication code using SHA-1.
- digest.hmac_sha256_base64() (/vcl/functions/digest-hmac-sha256-base64/) — Hash-based message authentication code using SHA-256.
- digest.hmac_sha256() (/vcl/functions/digest-hmac-sha256/) — Hash-based message authentication code using SHA-256.

- digest.hmac_sha512_base64() (/vcl/functions/digest-hmac-sha512-base64/) — Hash-based message authentication code using SHA-512.
- digest.hmac_sha512() (/vcl/functions/digest-hmac-sha512/) — Hash-based message authentication code using SHA-512.
- digest.rsa_verify() (/vcl/functions/digest-rsa-verify/) — A boolean function that returns true if the RSA signature of `payload` using `public_key` matches `digest`.
- digest.secure_is_equal() (/vcl/functions/digest-secure-is-equal/) — A boolean function that returns true if s1 and s2 are equal.
- digest.time_hmac_md5() (/vcl/functions/digest-time-hmac-md5/) — Returns a time-based one-time password using MD5 based upon the current time.
- digest.time_hmac_sha1() (/vcl/functions/digest-time-hmac-sha1/) — Returns a time-based one-time password using SHA-1 based upon the current time.
- digest.time_hmac_sha256() (/vcl/functions/digest-time-hmac-sha256/) — Returns a time-based one-time password with SHA-256 based upon the current time.
- digest.time_hmac_sha512() (/vcl/functions/digest-time-hmac-sha512/) — Returns a time-based one-time password with SHA-512 based upon the current time.

# Date and time (/vcl/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() (/vcl/functions/parse-time-delta/) — Parses a string representing a time delta and returns an integer number of seconds.
- std.integer2time() (/vcl/functions/std-integer2time/) — Converts an integer, representing seconds since the UNIX Epoch, to a time variable.
- std.time() (/vcl/functions/std-time/) — Converts a string to a time variable.
- strftime() (/vcl/functions/strftime/) — Formats a time to a string.
- time.add() (/vcl/functions/time-add/) — Adds a relative time to a time.
- time.hex_to_time() (/vcl/functions/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() (/vcl/functions/time-is-after/) — Returns true if `t1` is after `t2`.
- time.sub() (/vcl/functions/time-sub/) — Subtracts a relative time from a time.

# Miscellaneous (/vcl/miscellaneous/)

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

- addr.extract_bits() (/vcl/functions/addr-extract-bits/) — Extract `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() (/vcl/functions/addr-is-ipv4/) — Return true if the address family of the given address is IPv4.
- addr.is_ipv6() (/vcl/functions/addr-is-ipv6/) — Return true if the address family of the given address is IPv6.
- cstr_escape() (/vcl/functions/cstr-escape/) — Escapes bytes unsafe for printing from a string using C-style escape sequences.
- http_status_matches() (/vcl/functions/http-status-matches/) — Determines whether or not an HTTP status code matches a pattern.
- if() (/vcl/functions/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`.
- json.escape() (/vcl/functions/json-escape/) — Escapes characters of a UTF-8 encoded Unicode string using JSON-style escape sequences.
- regsub() (/vcl/functions/regsub/) — Replaces the first occurrence of `pattern`, which may be a Perl-compatible regular expression, in `input` with `replacement`.
- regsuball() (/vcl/functions/regsuball/) — Replaces all occurrences of `pattern`, which may be a Perl-compatible regular expression, in `input` with `replacement`.
- setcookie.get_value_by_name() (/vcl/functions/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.anystr2ip() (/vcl/functions/std-anystr2ip/) — Converts the string `addr` to an IP address (IPv4 or IPv6).
- std.atoi() (/vcl/functions/std-atoi/) — Takes a string (which represents an integer) as an argument and returns its value.
- std.collect() (/vcl/functions/std-collect/) — Combine multiple instances of the same header into one.
- std.ip() (/vcl/functions/std-ip/) — An alias of `std.str2ip()`.
- std.ip2str() (/vcl/functions/std-ip2str/) — Converts the IP address (v4 or v6) to a string.

- std.prefixof() (/vcl/functions/std-prefixof/) — True if the string `s` begins with the string `begins_with`.
- std.str2ip() (/vcl/functions/std-str2ip/) — Converts the string representation of an IP address (IPv4 or IPv6) into an `IP type`.
- std.strlen() (/vcl/functions/std-strlen/) — Returns the length of the string.
- std.strstr() (/vcl/functions/std-strstr/) — Returns the part of `haystack` string starting from and including the first occurrence of `needle` until the end of `haystack`.
- std.strtol() (/vcl/functions/std-strtol/) — Converts a string to an integer, using the second argument as base.
- std.suffixof() (/vcl/functions/std-suffixof/) — True if the string `s` ends with the string `ends_with`.
- std.tolower() (/vcl/functions/std-tolower/) — Changes the case of a string to lowercase.
- std.toupper() (/vcl/functions/std-toupper/) — Changes the case of a string to upper case.
- subfield() (/vcl/functions/subfield/) — Provides a means to access subfields from a header like `Cache-Control`, `Cookie`, and `Edge-Control` or individual parameters from the query string.
- urldecode() (/vcl/functions/urldecode/) — Decodes a percent-encoded string.
- urlencode() (/vcl/functions/urlencode/) — Encodes a string for use in a URL.

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

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

- boltsort.sort() (/vcl/functions/boltsort-sort/) — Sorts URL parameters.
- querystring.add() (/vcl/functions/querystring-add/) — Returns the given URL with the given parameter name and value appended to the end of the query string.
- querystring.clean() (/vcl/functions/querystring-clean/) — Returns the given URL without empty parameters.
- querystring.filter_except() (/vcl/functions/querystring-filter-except/) — Returns the given URL but only keeps the listed parameters.
- querystring.filter() (/vcl/functions/querystring-filter/) — Returns the given URL without the listed parameters.

- querystring.filtersep() (/vcl/functions/querystring-filtersep/) — Returns the separator needed by the `querystring.filter()` and `querystring.filter_except()` functions.
- querystring.globfilter_except() (/vcl/functions/querystring-globfilter-except/) — Returns the given URL but only keeps the parameters matching a glob.
- querystring.globfilter() (/vcl/functions/querystring-globfilter/) — Returns the given URL without the parameters matching a glob.
- querystring.regfilter_except() (/vcl/functions/querystring-regfilter-except/) — Returns the given URL but only keeps the parameters matching a regular expression.
- querystring.regfilter() (/vcl/functions/querystring-regfilter/) — Returns the given URL without the parameters matching a regular expression.
- querystring.remove() (/vcl/functions/querystring-remove/) — Returns the given URL with its query-string removed.
- querystring.set() (/vcl/functions/querystring-set/) — Returns the given URL with the given parameter name set to the given value, replacing the original value and removing any duplicates.
- querystring.sort() (/vcl/functions/querystring-sort/) — Returns the given URL with its query-string sorted.

# Randomness (/vcl/randomness/)

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

- randombool_seeded() (/vcl/functions/randombool-seeded/) — Identical to randombool, except takes an additional parameter, which is used to seed the random number generator.
- randombool() (/vcl/functions/randombool/) — Returns a random, boolean value.
- randomint_seeded() (/vcl/functions/randomint-seeded/) — Identical to randomint, except takes an additional parameter used to seed the random number generator.
- randomint() (/vcl/functions/randomint/) — Returns a random integer value between `from` and `to`, inclusive.
- randomstr() (/vcl/functions/randomstr/) — Returns a random string of length `len` containing characters from the supplied string `characters`.

# Table (/vcl/table/)

Tables provide a means to declare a constant dictionary and to efficiently look up values in the dictionary.

- table.lookup() (/vcl/functions/table-lookup/) — Look up the key `key` in the table `ID`.

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

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

- h2.disable_header_compression() (/vcl/functions/h2-disable-header-compression/) — Sets a flag to disable HTTP/2 header compression on one or many response headers to the client.
- h2.push() (/vcl/functions/h2-push/) — Triggers an HTTP/2 server push of the asset passed into the function as the input-string.

# UUID (/vcl/uuid/)

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

- uuid.dns() (/vcl/functions/uuid-dns/) — Returns the RFC4122 identifier of DNS namespace, namely the constant `"6ba7b810-9dad-11d1-80b4-00c04fd430c8"`.
- uuid.is_valid() (/vcl/functions/uuid-is-valid/) — Returns true if the string holds a textual representation of a valid UUID (per RFC4122).
- uuid.is_version3() (/vcl/functions/uuid-is-version3/) — Returns true if string holds a textual representation of a valid version 3 UUID.
- uuid.is_version4() (/vcl/functions/uuid-is-version4/) — Returns true if string holds a textual representation of a valid version 4 UUID.
- uuid.is_version5() (/vcl/functions/uuid-is-version5/) — Returns true if string holds a textual representation of a valid version 5 UUID.
- uuid.oid() (/vcl/functions/uuid-oid/) — Returns the RFC4122 identifier of ISO OID namespace, namely the constant `"6ba7b812-9dad-11d1-80b4-00c04fd430c8"`.
- uuid.url() (/vcl/functions/uuid-url/) — Returns the RFC4122 identifier of URL namespace, namely the constant `"6ba7b811-9dad-11d1-80b4-00c04fd430c8"`.
- uuid.version3() (/vcl/functions/uuid-version3/) — Derives a UUID corresponding to `name` within the given `namespace` using MD5 hash function.
- uuid.version4() (/vcl/functions/uuid-version4/) — Returns a UUID based on random number generator output.
- uuid.version5() (/vcl/functions/uuid-version5/) — Derives a UUID corresponding to `name` within the given `namespace` using SHA-1 hash function.
- uuid.x500() (/vcl/functions/uuid-x500/) — Returns the RFC4122 identifier of X.500 namespace, namely the constant `"6ba7b812-9dad-11d1-80b4-00c04fd430c8"`.

# Variables (/vcl/variables/)

These VCL variables are supported by Fastly.

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

By default VCL includes the `now` variable, which provides the current time (for example, `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 (/vcl/variables/now-sec/) — Like the `now` variable, but in seconds since the UNIX Epoch.
- now (/vcl/variables/now/) — The current time in RFC 1123 format (e.g., `Mon, 02 Jan 2006 22:04:05 GMT`).
- time.elapsed.msec_frac (/vcl/variables/time-elapsed-msec-frac/) — The time that has elapsed in milliseconds since the request started.
- time.elapsed.msec (/vcl/variables/time-elapsed-msec/) — The time since the request start in milliseconds.
- time.elapsed.sec (/vcl/variables/time-elapsed-sec/) — The time since the request start in seconds.
- time.elapsed.usec_frac (/vcl/variables/time-elapsed-usec-frac/) — The time the request started in microseconds since the last whole second.
- time.elapsed.usec (/vcl/variables/time-elapsed-usec/) — The time since the request start in microseconds.
- time.elapsed (/vcl/variables/time-elapsed/) — The time since the request started.
- time.end.msec_frac (/vcl/variables/time-end-msec-frac/) — The time the request started in milliseconds since the last whole second.
- time.end.msec (/vcl/variables/time-end-msec/) — The time the request ended in milliseconds since the UNIX Epoch.
- time.end.sec (/vcl/variables/time-end-sec/) — The time the request ended in seconds since the UNIX Epoch.
- time.end.usec_frac (/vcl/variables/time-end-usec-frac/) — The time the request started in microseconds since the last whole second.
- time.end.usec (/vcl/variables/time-end-usec/) — The time the request ended in microseconds since the UNIX Epoch.

- time.end (/vcl/variables/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 (/vcl/variables/time-start-msec-frac/) — The time the request started in milliseconds since the last whole second, after TLS termination.
- time.start.msec (/vcl/variables/time-start-msec/) — The time the request started in milliseconds since the UNIX Epoch, after TLS termination.
- time.start.sec (/vcl/variables/time-start-sec/) — The time the request started in seconds since the UNIX Epoch, after TLS termination.
- time.start.usec_frac (/vcl/variables/time-start-usec-frac/) — The time the request started in microseconds since the last whole second, after TLS termination.
- time.start.usec (/vcl/variables/time-start-usec/) — The time the request started in microseconds since the UNIX Epoch, after TLS termination.
- time.start (/vcl/variables/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 (/vcl/variables/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) (/vcl/esi/)

Fastly exposes tools to allow you to track a request that has ESI.

- req.esi (/vcl/variables/req-esi/) — Whether or not to enable ESI processing during this request.
- req.topurl (/vcl/variables/req-topurl/) — In an ESI subrequest, contains the URL of the top-level request.

# Geolocation (/vcl/geolocation/)

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

- client.as.name (/vcl/variables/client-as-name/) — The name of the organization associated with `client.as.number`.
- client.as.number (/vcl/variables/client-as-number/) — Autonomous system (AS) number.
- client.geo.area_code (/vcl/variables/client-geo-area-code/) — The telephone area code associated with the IP address.
- client.geo.city.ascii (/vcl/variables/client-geo-city-ascii/) — City or town name, encoded using ASCII encoding.

- client.geo.city.latin1 (/vcl/variables/client-geo-city-latin1/) — City or town name, encoded using Latin-1 encoding.
- client.geo.city.utf8 (/vcl/variables/client-geo-city-utf8/) — City or town name, encoded using UTF-8 encoding.
- client.geo.city (/vcl/variables/client-geo-city/) — Alias of `client.geo.city.ascii`.
- client.geo.conn_speed (/vcl/variables/client-geo-conn-speed/) — Connection speed.
- client.geo.continent_code (/vcl/variables/client-geo-continent-code/) — Two-letter code representing the continent.
- client.geo.country_code (/vcl/variables/client-geo-country-code/) — A two-character ISO 3166-1 country code for the country associated with the IP address.
- client.geo.country_code3 (/vcl/variables/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 (/vcl/variables/client-geo-country-name-ascii/) — Country name, encoded using ASCII encoding.
- client.geo.country_name.latin1 (/vcl/variables/client-geo-country-name-latin1/) — Country name, encoded using Latin-1 encoding.
- client.geo.country_name.utf8 (/vcl/variables/client-geo-country-name-utf8/) — Country name, encoded using UTF-8 encoding.
- client.geo.country_name (/vcl/variables/client-geo-country-name/) — Alias of `client.geo.country_name.ascii`.
- client.geo.gmt_offset (/vcl/variables/client-geo-gmt-offset/) — Time zone offset from coordinated universal time (UTC) for `client.geo.city`.
- client.geo.ip_override (/vcl/variables/client-geo-ip-override/) — Override the IP address for geolocation data.
- client.geo.latitude (/vcl/variables/client-geo-latitude/) — Latitude, in units of degrees from the equator.
- client.geo.longitude (/vcl/variables/client-geo-longitude/) — Longitude, in units of degrees from the IERS Reference Meridian.
- client.geo.metro_code (/vcl/variables/client-geo-metro-code/) — Metro code.
- client.geo.postal_code (/vcl/variables/client-geo-postal-code/) — The postal code associated with the IP address.
- client.geo.region.ascii (/vcl/variables/client-geo-region-ascii/) — ISO 3166-2 country subdivision code.

- client.geo.region.latin1 (/vcl/variables/client-geo-region-latin1/) — Region code, encoded using Latin-1 encoding.
- client.geo.region.utf8 (/vcl/variables/client-geo-region-utf8/) — Region code, encoded using UTF-8 encoding.
- client.geo.region (/vcl/variables/client-geo-region/) — Alias of `client.geo.region.ascii`.

# Miscellaneous (/vcl/miscellaneous/)

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

- bereq.url.basename (/vcl/variables/bereq-url-basename/) — Same as `req.url.basename`, except for use between Fastly and your origin servers.
- bereq.url.dirname (/vcl/variables/bereq-url-dirname/) — Same as `req.url.dirname`, except for use between Fastly and your origin servers.
- bereq.url.qs (/vcl/variables/bereq-url-qs/) — The query string portion of `bereq.url`.
- bereq.url (/vcl/variables/bereq-url/) — The URL sent to the backend.
- beresp.backend.ip (/vcl/variables/beresp-backend-ip/) — The IP of the backend this response was fetched from (backported from Varnish 3).
- beresp.backend.name (/vcl/variables/beresp-backend-name/) — The name of the backend this response was fetched from (backported from Varnish 3).
- beresp.backend.port (/vcl/variables/beresp-backend-port/) — The port of the backend this response was fetched from (backported from Varnish 3).
- beresp.grace (/vcl/variables/beresp-grace/) — Defines how long an object can remain overdue and still have Varnish consider it for grace mode.
- beresp.hipaa (/vcl/variables/beresp-hipaa/) — Specifies that content not be cached in non-volatile memory to help customers meet HIPAA security requirements.
- beresp.pci (/vcl/variables/beresp-pci/) — Specifies that content be cached in a manner that satisfies PCI DSS requirements.
- client.ip (/vcl/variables/client-ip/) — The IP address of the client making the request.
- client.port (/vcl/variables/client-port/) — Returns the remote client port.
- client.requests (/vcl/variables/client-requests/) — Tracks the number of requests received by Varnish over a persistent connection.
- client.socket.pace (/vcl/variables/client-socket-pace/) — Ceiling rate in kilobytes per second for bytes sent to the client.

- req.body.base64 (/vcl/variables/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 (/vcl/variables/req-body/) — The request body.
- req.grace (/vcl/variables/req-grace/) — Defines how long an object can remain overdue and still have Varnish consider it for grace mode.
- req.http.host (/vcl/variables/req-http-host/) — The full host name, without the path or query parameters.
- req.is_ipv6 (/vcl/variables/req-is-ipv6/) — Indicates whether the request was made using IPv6 or not.
- req.restarts (/vcl/variables/req-restarts/) — Counts the number of times the VCL has been restarted.
- req.url.basename (/vcl/variables/req-url-basename/) — The file name specified in a URL.
- req.url.dirname (/vcl/variables/req-url-dirname/) — The directories specified in a URL.
- req.url.ext (/vcl/variables/req-url-ext/) — The file extension specified in a URL.
- req.url.path (/vcl/variables/req-url-path/) — The full path, without any query parameters.
- req.url.qs (/vcl/variables/req-url-qs/) — The query string portion of `req.url`.
- req.url (/vcl/variables/req-url/) — The full path, including query parameters.
- stale.exists (/vcl/variables/stale-exists/) — Specifies if a given object has stale content in cache.

# Server (/vcl/server/)

Variables relating to the server receiving the request.

- server.datacenter (/vcl/variables/server-datacenter/) — A code representing one of Fastly's POP locations.
- server.hostname (/vcl/variables/server-hostname/) — Hostname of the server (e.g., `cache-jfk1034`).
- server.identity (/vcl/variables/server-identity/) — Same as `server.hostname` but also explicitly includes the datacenter name (e.g., `cache-jfk1034-JFK`).
- server.region (/vcl/variables/server-region/) — A code representing the general region of the world in which the POP location resides.

# Size (/vcl/size/)

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

- beresp.body_bytes_written (/vcl/variables/beresp-body-bytes-written/) — Total body bytes written to a backend.
- beresp.header_bytes_written (/vcl/variables/beresp-header-bytes-written/) — Total header bytes written to a backend.
- req.body_bytes_read (/vcl/variables/req-body-bytes-read/) — Total body bytes read from the client generating the request.
- req.bytes_read (/vcl/variables/req-bytes-read/) — Total bytes read from the client generating the request.
- req.header_bytes_read (/vcl/variables/req-header-bytes-read/) — Total header bytes read from the client generating the request.
- resp.body_bytes_written (/vcl/variables/resp-body-bytes-written/) — Body bytes to send to the client in the response.
- resp.bytes_written (/vcl/variables/resp-bytes-written/) — Total bytes to send to the client in the response.
- resp.completed (/vcl/variables/resp-completed/) — Whether the response completed successfully or not.
- resp.header_bytes_written (/vcl/variables/resp-header-bytes-written/) — How many bytes were written for the header of a response.

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

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

- fastly_info.h2.is_push (/vcl/variables/fastly-info-h2-is-push/) — Whether or not this request was a server-initiated request generated to create an HTTP/2 Server-pushed response.
- fastly_info.h2.stream_id (/vcl/variables/fastly-info-h2-stream-id/) — If the request was made over HTTP/2, the underlying HTTP/2 stream ID.
- fastly_info.is_h2 (/vcl/variables/fastly-info-is-h2/) — Whether or not the request was made using http2.
- tls.client.cipher (/vcl/variables/tls-client-cipher/) — The cipher suite used to secure the client TLS connection.
- tls.client.ciphers_list_sha (/vcl/variables/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 (/vcl/variables/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 (/vcl/variables/tls-client-ciphers-list/) — The list of ciphers supported by the client, as sent over the network, hex encoded.
- tls.client.ciphers_sha (/vcl/variables/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 (/vcl/variables/tls-client-protocol/) — The TLS protocol version this connection is speaking over.
- tls.client.servername (/vcl/variables/tls-client-servername/) — The Server Name Indication (SNI) the client sent in the `ClientHello` TLS record.
- tls.client.tlsexts_list_sha (/vcl/variables/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 (/vcl/variables/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 (/vcl/variables/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 (/vcl/variables/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 (/vcl/local-variables/)

Fastly VCL (/guides/vcl-tutorials/guide-to-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` (/vcl/types/bool/)
- `FLOAT` (/vcl/types/float/)
- `INTEGER` (/vcl/types/integer/)
- `IP` (/vcl/types/ip/)
- `RTIME` (/vcl/types/rtime/) (relative time)
- `STRING` (/vcl/types/string/)
- `TIME` (/vcl/types/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 (/vcl/operators/)

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

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

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

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

# Operator precedence

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

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

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

# Negation

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

```
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: $\infty - \infty$ and $0 \times \infty$. These evaluate to NaN but do not set `fastly.error`.

## Assignment

Assignment is provided by the `=` operator:

```
1  := "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

```
1  := "set" variable "|=" assignment-expr ";"
2   | "set" variable "&=" assignment-expr ";"
3   | "set" variable "^=" assignment-expr ";"
4   | "set" variable ">>=" assignment-expr ";"
5   | "set" variable "<<=" assignment-expr ";"
6   | "set" variable "ror=" assignment-expr ";"
7   | "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:

```
1  := "set" variable "&&=" assignment-expr ";"
2   | "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:

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

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

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

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

## Comparison operators

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

STRING comparisons have special cases for operands which are not set (as opposed to empty): The `!=` and `!~` operators always evaluate to true when either operand is not set. All other conditional operators always evaluate to false when either operand is not set. For example, if a given variable is not set, that variable will compare unequal to itself: both `req.http.unset == req.http.unset` and `req.http.unset ~ ".?"` will be false.

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

The comparison operators are:

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

Equality is defined for all types:

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

Inequalities are defined for numeric types and TIME:

```
1  := numeric-expr lg-op numeric-expr
2   | 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/types/)

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

## Types for scalar values

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

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

- BOOL (/vcl/types/bool/)
- FLOAT (/vcl/types/float/)
- INTEGER (/vcl/types/integer/)

- IP (/vcl/types/ip/)
- RTIME (/vcl/types/rtime/)
- STRING (/vcl/types/string/)
- TIME (/vcl/types/time/)

## Types with special semantics

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

- BACKEND (/vcl/types/backend/)
- HASH (/vcl/types/hash/)
- HEADER (/vcl/types/header/)
- VOID (/vcl/types/void/)

# Directors (/vcl/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

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.

In the following example, the fallback director will send all requests to `F_backend1`, until its health status is unhealthy. If `F_backend1` becomes unhealthy, the fallback director will send all requests to `F_backend2` until `F_backend1` is healthy again. If `F_backend1` and `F_backend2` both become unhealthy, the fallback director will send all requests to `F_backend3` until either one of the previous backends become healthy again.

```
director my_dir fallback {
  { .backend = F_backend1; }
  { .backend = F_backend2; }
  { .backend = F_backend3; }
}
```