Content negotiation

Content negotiation Functions

accept.charset_lookup()

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

This function takes the following parameters:

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

Format

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

Examples

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

accept.encoding_lookup()

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

This function takes the following parameters:

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

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

Format

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

Examples

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

accept.language_filter_basic()

Similar to <u>accept.language_lookup()</u>, this function selects the best matches from a string in the format of an <u>Accept-Language</u> h languages, using the algorithm described in <u>RFC 4647</u>, Section 3.3.1.

This function takes the following parameters:

1. a colon-separated list of languages available for the resource,

- 2. a fallback return value,
- 3. a string representing an Accept-Language header's value,
- 4. the maximum number of matching languages to return.

The matches are comma-separated.

Format

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

Examples

```
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()

Selects the best match from a string in the format of an Accept-Language header's value in the listed languages, using the algorithr 3.4.

This function takes the following parameters:

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

This function conforms to RFC 4647.

Format

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

Examples

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

accept.media_lookup()

Selects the best match from a string in the format of an Accept header's value in the listed media types, using the algorithm descrik

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. Value not contain variables. Duplicate media types among the first three arguments are not allowed.

Format

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

Examples

```
1  # We wish `image/jpeg` to return `image/jpeg`.
2  # We wish `image/png` to return `image/png`.
3  # We wish `image/* to return `image/tiff`.
4  # We wish `text/*` to return `text/html`.
5  # We wish `*/*` to return `text/plain`.
6  set beresp.http.media = accept.media_lookup("image/jpeg:image/png",
7  "text/plain",
8  "image/tiff:text/html",
9  req.http.Accept);
```

Cryptographic

Notes

In Base64 decoding, the output theoretically could be in binary but is interpreted as a string. So if the binary output contains '\0' there are time based One-Time Password algorithm initializes the HMAC using the key and appropriate hash type. Then it hashes the mes

```
(<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?6h2YU11CB4C50SbkZ0E6U3dZGjh+84dz3+Zope2Uhik=
```

Example implementations for token generation in various languages can be found in GitHub.

Example VCL

```
sub vcl_recv {
 1
 2
 3
      /* make sure there is a token */
      if (req.url !~ "[?&]token=([^&]+)") {
 4
 5
        error 403;
 6
      }
 7
 8
      if (re.group.1 != digest.time_hmac_sha256("RmFzdGx5IFRva2VuIFRlc3Q=", 60, 0) &&
          re.group.1 != digest.time_hmac_sha256("RmFzdGx5IFRva2VuIFRlc3Q=", 60, -1)) {
 9
10
        error 403;
      }
11
12
13 #FASTLY recv
14
15
16
   }
```

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 Base6.

```
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 to you intend to send a synthetic response that contains binary data. There is currently no way to send a synthetic response containing

Cryptographic Functions

digest.awsv4 hmac()

Returns an <u>AWSv4 message authentication code</u> based on the supplied key and string. This function automatically prepends "Alkey (the first function parameter) as required by the protocol. This function does not support binary data for its key or string parameter.

Format

```
STRING
digest.awsv4_hmac(STRING key, STRING date_stamp, STRING region, STRING service, STRING string)
```

Examples

```
1 declare local var.signature STRING;
2 set var.signature = digest.awsv4_hmac(
3    "wJalrXUtnFEMI/K7MDENG+bPxRfiCYEXAMPLEKEY",
4    "20120215",
5    "us-east-1",
6    "iam",
7    "hello");
```

digest.base64_decode()

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

Format

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

Examples

```
1 declare local var.base64_decoded STRING;
2 set var.base64_decoded = digest.base64_decode("zpr0sc67z47PgiD0v8+Bzq/Pg86xz4T0tQ==");
3 # var.base64_decoded is now "Καλώς ορίσατε"
```

digest.base64()

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

Format

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

Examples

```
1 declare local var.base64_encoded STRING;
2 set var.base64_encoded = digest.base64("Καλώς ορίσατε");
3 # var.base64_encoded is now "zpr0sc67z47PgiD0v8+Bzq/Pg86xz4T0tQ=="
```

digest.base64url_decode()

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

Format

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

Examples

```
1 declare local var.base64url_decoded STRING;
2 set var.base64url_decoded = digest.base64url_decode("zpr0sc67z47PgiD0v8-Bzq_Pg86xz4T0tQ==");
3 # var.base64url_decoded is now "Καλώς ορίσατε"
```

digest.base64url nopad decode()

Returns the Base64 decoding with URL and filename safe alphabet decoding of the input string, as specified by RFC 4648, without I

Format

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

Examples

```
1 declare local var.base64url_nopad_decoded STRING;
2 set var.base64url_nopad_decoded = digest.base64url_nopad_decode("zpr0sc67z47PgiD0v8-Bzq_Pg86xz4T0tQ");
3 # var.base64url_nopad_decoded is now "Καλώς ορίσατε"
```

digest.base64url_nopad()

Returns the Base64 encoding with URL and filename safe alphabet encoding of the input string, as specified by RFC 4648, without processes the same safe alphabet encoding of the input string, as specified by RFC 4648, without processes and the safe alphabet encoding of the input string, as specified by RFC 4648, without processes are safe alphabet encoding of the input string, as specified by RFC 4648, without processes are safe alphabet encoding of the input string, as specified by RFC 4648, without processes are safe alphabet encoding of the input string.

Format

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

Examples

```
1 declare local var.base64url_nopad_encoded STRING;
2 set var.base64url_nopad_encoded = digest.base64url_nopad("Καλώς ορίσατε");
3 # var.base64url_nopad_encoded is now "zpr0sc67z47PgiD0v8-Bzq_Pg86xz4TOtQ"
```

digest.base64url()

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

Format

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

Examples

```
1 declare local var.base64url_encoded STRING;
2 set var.base64url_encoded = digest.base64url("Καλώς ορίσατε");
3 # var.base64url_encoded is now "zpr0sc67z47PgiD0v8-Bzq_Pg86xz4TOtQ=="
```

digest.hash_crc32()

Calculates the 32-bit Cyclic Redundancy Checksum with reversed bit ordering of a string, like that used by <u>bzip2</u>. Returns a hex-enc e.g. [181989fc] instead of [fc891918].

Format

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

Examples

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

digest.hash_crc32b()

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

Format

```
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()

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

Format

```
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()

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

Format

```
STRING
digest.hash_shal(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()

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

Format

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

Examples

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

digest.hash_sha256()

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

Format

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

Examples

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

digest.hash_sha384()

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

Format

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

Examples

```
1 declare local var.hash_sha384 STRING;
2 set var.hash_sha384 = digest.hash_sha384("123456789");
3 # var.hash_sha384 is now "eb455d56d2c1a69de64e832011f3393d45f3fa31d6842f21af92d2fe469c499da5e3179847334a18479c8d1ded
```

digest.hash_sha512()

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

Format

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

Examples

```
1 declare local var.hash_sha512 STRING;
2 set var.hash_sha512 = digest.hash_sha512("123456789");
3 # var.hash_sha512 is now "d9e6762dd1c8eaf6d61b3c6192fc408d4d6d5f1176d0c29169bc24e71c3f274ad27fcd5811b313d681f7e55ec0 a8ffe85"
```

digest.hmac md5 base64()

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

Format

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

Examples

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

digest.hmac_md5()

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

Format

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

Examples

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

digest.hmac_sha1_base64()

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

Format

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

Examples

```
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 "hR07NVB2z0KuXrnzmatcr9unyKI="
```

digest.hmac_sha1()

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

Format

```
STRING
digest.hmac_shal(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()

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

Format

```
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/Y502g6cuM="
```

digest.hmac sha256()

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

Format

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

Examples

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

digest.hmac_sha512_base64()

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

Format

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

Examples

```
1 declare local var.hmac_sha512_base64 STRING;
2 set var.hmac_sha512_base64 = digest.hmac_sha512_base64("key", "input");
3 # var.hmac_sha512_base64 is now "A613yBfzJmnMzzjayRXU5VoWgzscpoWXmp31IaBSNZeAkAQ8PaQC2tNl7TmsBa9IZKgERRhh9LTfdoCDTG1
```

digest.hmac_sha512()

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

Format

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

Examples

```
1 declare local var.hmac_sha512 STRING;
2 set var.hmac_sha512 = digest.hmac_sha512("key", "input");
3 # var.hmac_sha512 is now "0x03ad77c817f32669cccf38dac915d4e55a16833b1ca685979a9df521a05235978090043c3da402dad365ed3934c6d4f95"
```

digest.rsa_verify()

A boolean function that returns true if the RSA signature of <code>payload</code> using <code>public_key</code> matches <code>digest</code>. The <code>hash_method</code> paramuse. It can be <code>sha256</code>, <code>sha384</code>, <code>sha512</code>, or <code>default</code> (<code>default</code> is equivalent to <code>sha256</code>). The <code>string_list</code> parameter in the paylous such as <code>req.http.payload</code> and <code>req.http.digest</code>. The <code>base64_method</code> parameter is optional. It can be <code>standard</code>, <code>url</code>, <code>url_nogential control of the standard</code> and <code>req.http.digest</code>. The <code>base64_method</code> parameter is optional. It can be <code>standard</code>, <code>url</code>, <code>url_nogential control of the standard</code> and <code>standard</code>.

Format

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

Examples

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

digest.secure is equal()

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

Format

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

Examples

```
if (!(table.lookup(user2hashedpass, req.http.User) && digest.secure_is_equal(req.http.HashedPass, table.lookup(user2 error 401 "Unauthorized";
}
```

digest.time hmac md5()

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

Format

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

Examples

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

digest.time hmac sha1()

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

Format

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

Examples

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

digest.time hmac sha256()

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

Format

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

Examples

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

digest.time hmac sha512()

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

Format

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

Examples

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

Date and time

Date and time Functions



parse time delta()

Parses a string representing a time delta and returns an integer number of seconds. This function supports the specifiers "d" and "D "m" and "M" for minutes, and "s" and "S" for seconds. The function parses individual deltas like "15m" and "7d". Strings like "10d11 of invalid input, the function returns -1.

Format

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

Examples

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



std.integer2time()

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

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

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

Format

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

Examples

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

std.time()

Converts a string to a time variable.

The following string formats are supported:

- Mon, 02 Jan 2006 22:04:05 GMT, RFC 822 and RFC 1123
- Monday, 02-Jan-06 22:04:05 GMT, RFC 850
- Mon Jan 2 22:04:05 2006, ANSI-C asctime()
- 2006-01-02 22:04:05, an ISO 8601 subset
- 1136239445.00, seconds since the UNIX Epoch
- 1136239445, seconds since the UNIX Epoch

The only time zone supported is GMT.

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

Format

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

Examples

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

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

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

strftime()

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

 \bigstar TIP: Regular strings ("short strings") in VCL use \S_{xx} escapes (percent encoding) for special characters, which would conflict v format. For the strftime examples, we use VCL "long strings" $\{"\dots"\}$, which do not use the \S_{xx} escapes. Alternatively, you could be used to the \S_{xx} escapes.

Format

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

Examples

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

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

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

time.add()

Adds a relative time to a time.

Format

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

Examples

```
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()

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

Format

```
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()

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

Format

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

Examples

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

time.sub()

Subtracts a relative time from a time.

Format

```
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



Like the <u>now</u> variable, but in seconds since the <u>UNIX Epoch</u>.

Type

STRING

Accessibility

Readable From

All subroutines



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

Type

TIME

Accessibility

Readable From



time.elapsed.msec frac

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

Type

STRING

Accessibility

Readable From

All subroutines



time.elapsed.msec

The time since the request start in milliseconds.

Type

STRING

Accessibility

Readable From

All subroutines



itime.elapsed.sec

The time since the request start in seconds.

Type

STRING

Accessibility

Readable From

All subroutines



time.elapsed.usec_frac

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

Type

STRING

Accessibility

Readable From

All subroutines



time.elapsed.usec

The time since the request start in microseconds.

Type

STRING

Accessibility

Readable From

All subroutines



time.elapsed

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

Type

RTIME

Accessibility

Readable From

All subroutines



time.end.msec_frac

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

Type

STRING

Accessibility

Readable From

- vcl_deliver
- vcl_log



time.end.msec

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

Type

STRING

Accessibility

Readable From

- vcl_deliver
- vcl_log



time.end.sec

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

Type

STRING

Accessibility

Readable From

- vcl deliver
- vcl_log



time.end.usec_frac

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

Type

STRING

Accessibility

Readable From

- vcl deliver
- vcl_log

time.end.usec

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

Type

STRING

Accessibility

Readable From

- vcl deliver
- vcl_log



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

Type

TIME

Accessibility

Readable From

- vcl_deliver
- vcl_log



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

Type

STRING

Accessibility

Readable From

All subroutines



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

Type

STRING

Accessibility

Readable From

All subroutines



The time the request started in seconds since the <u>UNIX Epoch</u>, after TLS termination.

Type

STRING

Accessibility

Readable From

All subroutines



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

Type

STRING

Accessibility

Readable From

All subroutines



time.start.usec

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

Type

STRING

Accessibility

Readable From

All subroutines



time.start

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

Type

TIME

Accessibility

Readable From

All subroutines



itime.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 R formatted as a number in seconds with 3 decimal digits of precision. In vcl deliver this interval will be very close to time.elapse between time.elapsed and time.to_first_byte will be the time that it took to send the response body.

Type

RTIME

Accessibility

Readable From

- vcl deliver
- vcl log

Edge Side Includes (ESI)

Edge Side Includes (ESI) Variables



req.esi

Whether or not to disable or enable ESI processing during this request. Using set req.esi = false; will disable ESI processing. T

Type

BOOL

Accessibility

Readable From

- vcl_recv
- vcl fetch
- vcl_deliver
- vcl_error



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

Type

STRING

Accessibility

Readable From

All subroutines

Floating point classification

Floating point values are grouped into one of several classifications:

• Finite — math.is finite()

A value that is neither NaN nor an infinity.

• Subnormal - math.is_subnormal()

The **FLOAT** type supports <u>subnormals</u> (also known as *denormals*).

• NaN - math.is nan()

The **FLOAT** type may express NaN (Not a Number). In general, arithmetic operations involving a NaN will produce NaN. NaN va **fastly.error** variable.

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

• Normal - math.is normal()

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

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

• Infinite — math.is infinite()

The FLOAT type may express IEEE 754 *infinities*. These are signed values. Infinities behave with special semantics for some or There is no literal syntax for assigning infinities, but math.POS INFINITY and math.NEG INFINITY constants are provided.

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

No VCL function is provided to determine whether a floating point value is a zero. Because both positive and negative zero con made simply by var.x == 0.

Floating point classification Functions

math.is_finite()

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

Format

BOOL
math.is_finite(FLOAT x)

math.is_infinite()

Determines whether a floating point value is an infinity. See <u>floating point classifications</u> for more information.

Format

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

Examples

```
1 declare local var.f FLOAT;
2
3 set var.f = math.POS_INFINITY;
4 set var.f -= 1; # +\infty - 1 produces +\infty
5 if (math.is_infinite(var.f)) {
6 log "infinity";
7 }
```

math.is_nan()

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

Format

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

Examples

```
1 declare local var.f FLOAT;
2
3 set var.f = 1;
4 set var.f /= 0;
5 if (math.is_nan(var.f)) {
6 log "division by zero";
7 }
```

math.is_normal()

Determines whether a floating point value is normal. See <u>floating point classifications</u> for more information.

Format

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

Examples

```
1 # zeroes are not normals
2 if (!math.is_normal(0)) {
3  log "not a normal";
4 }
```

math.is_subnormal()

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

Format

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

Examples

```
declare local var.f FLOAT;
 1
 2
 3
    set var.f = math.FLOAT_MIN; # minimum finite value
 4
    if (!math.is_subnormal(var.f)) {
      log "not subnormal";
 5
 6 }
 7
    set var.f /= 2;
   if (math.is_subnormal(var.f)) {
 8
 9
      log "subnormal";
10
```

Geolocation

All geographic data presented through these variables is associated with a particular IP address. This address is automatically popul 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. The removed and are normalized to lowercase spellings. These *.ascii variables can be used as a symbolic string in code (for example depending on the city name). Due to their simplified content, however, they are generally inappropriate for presenting to users.

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

1 NOTE: Geolocation information, including data streamed by our <u>log streaming service</u>, is intended to be used only in connectic services. Use of geolocation data for other purposes may require the permission of an IP geolocation dataset vendor, such as <u>Dig</u>

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

Using geographic variables with shielding

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

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

Absent data

▲ WARNING: The geolocation data is updated periodically as IP allocations change and various amendments are made. Some variety 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

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

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

Reserved IP address blocks

The IPv4 and IPv6 address spaces have several blocks reserved for special uses. These include <u>private use networks</u> (e.g., 192.168. address ranges reserved for documentation (e.g., 203.0.113.0/24 <u>RFC 5737</u> TEST-NET-3).

Geographic data has no meaningful association for these ranges. The geolocation VCL variables present special values for these ran

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

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

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

Type

STRING

Accessibility

Readable From

All subroutines



client.as.number

Autonomous system (AS) number.

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

Formatting these numbers to base 10 (e.g., by implicit conversion to a STRING type) will give an asplain representation of the num representation.

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

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

Examples

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

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

```
declare local var.as_category STRING;
 2 if (client.as.number < 0 || client.as.number > 0xFFFFFFFF) {
     set var.as_category = "invalid";
3
 4 } else if (client.as.number == 0) {
     set var.as_category = "reserved"; # RFC 1930
 5
 6 } else if (client.as.number <= 23455) {
7
     set var.as_category = "public";
 8 } else if (client.as.number == 23456) {
    set var.as_category = "transition"; # RFC 6793
 9
10 } else if (client.as.number <= 64534) {
11
     set var.as category = "public";
12 } else if (client.as.number <= 64495) {
    set var.as_category = "reserved";
14 } else if (client.as.number <= 64511) {</pre>
   set var.as_category = "documentation"; # RFC 5398
16 } else if (client.as.number <= 65534) {</pre>
17 set var.as_category = "private";
18  } else if (client.as.number == 65535) {
19
   set var.as_category = "reserved";
20  } else if (client.as.number <= 65551) {</pre>
21 set var.as_category = "documentation"; # RFC 4893, RFC 5398
22 } else if (client.as.number <= 131071) {
23 set var.as_category = "reserved";
24 } else if (client.as.number <= 4199999999) {
25
   set var.as_category = "public";
26 } else if (client.as.number <= 4294967294) {
   set var.as_category = "private"; # RFC 6996
27
28 } else if (client.as.number == 4294967295) {
   set var.as_category = "reserved";
29
30 } else {
    set var.as_category = "unknown";
31
```

Type

INTEGER

Accessibility

Readable From

All subroutines



client.geo.area_code

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

Type

INTEGER

Accessibility

Readable From

All subroutines



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

Type

STRING

Accessibility

Readable From

All subroutines



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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.city.utf8

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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.city

Alias of client.geo.city.ascii.

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.conn speed

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

Possible values are: broadband, cable, dialup, mobile, oc12, oc3, t1, t3, satellite, wireless, xds1.

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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.continent code

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

Code	Continent
AF	Africa
AN	Antarctica
AS	Asia
EU	Europe
NA	North America
ОС	Oceania

South America SA

These continents are defined by UN M.49.

The continent code for the Caribbean countries is NA.

Note that Eu refers to the continent name, not to the European Union. For example, IP addresses allocated to Norway and Switzerla Economic Area and the Schengen Area respectively, but not of the European Union) are presented with the continent code EU, mean Europe.

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.country_code

A two-character ISO 3166-1 country code for the country associated with the IP address. The US country code is returned for IP add United States military bases.

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

Examples

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

```
1
    table unicode_ri {
      "A": "%u{1F1E6}", "B": "%u{1F1E7}", "C": "%u{1F1E8}", "D": "%u{1F1E9}",
 2
      "E": "%u{1F1EA}", "F": "%u{1F1EB}", "G": "%u{1F1EC}", "H": "%u{1F1ED}",
 3
      "I": "%u{1F1EE}", "J": "%u{1F1EF}", "K": "%u{1F1F0}", "L": "%u{1F1F1}",
 4
      "M": "%u{1F1F2}", "N": "%u{1F1F3}", "O": "%u{1F1F4}", "P": "%u{1F1F5}",
 5
      "Q": "%u{1F1F6}", "R": "%u{1F1F7}", "S": "%u{1F1F8}", "T": "%u{1F1F9}",
 6
      "U": "%u{1F1FA}", "V": "%u{1F1FB}", "W": "%u{1F1FC}", "X": "%u{1F1FD}",
 7
      "Y": "%u{1F1FE}", "Z": "%u{1F1FF}"
 8
 9
    }
10
11
    set resp.http.X-flag = table.lookup(unicode_ri, substr(client.geo.country_code, 0, 1))
12
                            table.lookup(unicode_ri, substr(client.geo.country_code, 1, 1));
```

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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.country code3

A three-character ISO 3166-1 alpha-3 country code for the country associated with the IP address. The USA country code is returne overseas United States military bases.

Type

STRING

Accessibility

Readable From

All subroutines

client.geo.country_name.ascii

Country name, encoded using ASCII encoding.

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

Examples

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

Type

STRING

Accessibility

Readable From

All subroutines



i client.geo.country name.latin1

Country name, encoded using Latin-1 encoding.

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.country name.utf8

Country name, encoded using UTF-8 encoding.

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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.country name

Alias of client.geo.country_name.ascii.

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.gmt offset

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

1 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 hot -230 would be offset of minus two hours and thirty minutes from UTC.

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

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

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

Examples

Not all timezone offsets are on the hour. For example, in St. John's, Newfoundland, client.geo.gmt_offset may be -230 or -330 (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;
4 set var.offset_by_hour /= 60; # minutes
  set var.offset_by_hour += std.atoi(regsub(client.geo.gmt_offset, "..$", "")); # truncate
```

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

Type

INTEGER

Accessibility

Readable From

All subroutines



client.geo.ip override

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

It is possible to set client.geo.ip override to an invalid IP address:

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

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

Type

IP

Accessibility

Readable From

All subroutines



client.geo.latitude

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

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

Examples

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

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

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

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

```
1 declare local var.zone STRING;
2 if (client.geo.latitude == 999.9) {
     set var.zone = "";
3
 4 } else if (client.geo.latitude >= 66.5) { # Arctic circle
     set var.zone = "North frigid";
 5
 6 } else if (client.geo.latitude >= 23.5) { # Topic of Cancer
 7
     set var.zone = "North temperate";
 8 } else if (client.geo.latitude <= -66.5) { # Antarctic Circle
 9
     set var.zone = "South frigid";
10 } else if (client.geo.latitude <= -23.5) { # Tropic of Capricorn
     set var.zone = "South temperate";
11
12 } else {
13
     set var.zone = "Torrid";
14 }
```

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

```
1 declare local var.deg INTEGER;
 2 declare local var.min INTEGER;
 3 declare local var.sec FLOAT;
 4
 5 declare local var.angle FLOAT;
 6 declare local var.whole FLOAT;
7
   declare local var.frac FLOAT;
 8
 9 set var.angle = client.geo.latitude; # input
10
   if (var.angle < 0.0) {
     set var.angle *= -1;
11
12 }
13
14 set var.frac = var.angle;
15 set var.whole = var.frac;
16 set var.frac %= 1.0;
17
   set var.whole -= var.frac;
18
   set var.deg = var.whole; # truncated, integer by rounding
19
20 set var.frac *= 60.0;
   set var.whole = var.frac;
21
22
   set var.frac %= 1.0;
23
   set var.whole -= var.frac;
24
    set var.min = var.whole; # truncated, integer by rounding
25
26 set var.sec = var.frac;
27 set var.sec *= 60.0; # floating seconds
28
29 log client.geo.latitude + " = " + var.deg "° " var.min "′ " var.sec "″ "
     + if (client.geo.latitude < 0.0, "S", "N");
30
```

For example, a latitude of 59.926 produces [59° 55' 33.600" N]. The r and r symbols are Unicode prime symbols, not quotes.

Type

FLOAT

Accessibility

Readable From

All subroutines



i client.geo.longitude

Longitude, in units of degrees from the IERS Reference Meridian. Values range from -180 to +180 inclusive, with the exception of the

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

Type

FLOAT

Accessibility

Readable From

All subroutines



client.geo.metro code

Metro code.

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

INTEGER

Accessibility

Readable From

All subroutines



client.geo.postal code

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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.region.ascii

ISO 3166-2 country subdivision code. For countries with multiple levels of subdivision (for example, nations within the United Kingdo specific subdivision.

The special value NO REGION is given for countries that do not have ISO country subdivision codes. For example, NO REGION is give Aland 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 coc illustrates constructing an ISO 3166-2 two-part country and subdivision code from the respective variables:

```
1 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;
5
6
     }
  }
```

Examples

Here are some example values:

var.code	Region Name	Country	ISO 3166-2 subdivision
AX	Ödkarby	Åland Islands	(none)
DE-BE	Berlin	Germany	Land (State)
GB-BNH	Brighton and Hove	United Kingdom	Unitary authority
JP-13	東京都 (Tōkyō-to)	Japan	Prefecture

RU-MO	WC	Москва́ (Moscow)	Russian Federation	Federal city
SE-AF	3	Stockholms län	Sweden	Län (County)
US-CA	A	California	United States	State

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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.region.latin1

Region code, encoded using Latin-1 encoding.

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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.region.utf8

Region code, encoded using UTF-8 encoding.

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

Type

STRING

Accessibility

Readable From

All subroutines



client.geo.region

Alias of client.geo.region.ascii.

Type

STRING

Accessibility

Readable From

All subroutines

Math constants and limits

Math constants and limits Variables



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

Type

FLOAT

Accessibility

Readable From

All subroutines



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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.2 SQRTPI

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.2PI

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.E

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.FLOAT DIG

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

Type

INTEGER

Accessibility

Readable From



math.FLOAT EPSILON

Minimum positive difference from 1.0 for the FLOAT type.

Type

FLOAT

Accessibility

Readable From

All subroutines



math.FLOAT MANT DIG

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

Type

INTEGER

Accessibility

Readable From

All subroutines



math.FLOAT MAX 10 EXP

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

Type

INTEGER

Accessibility

Readable From

All subroutines



math.FLOAT MAX EXP

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

Type

INTEGER

Accessibility

Readable From

All subroutines



math.FLOAT MAX

Maximum finite value for the **FLOAT** type.

Type

FLOAT

Accessibility

Readable From

All subroutines



math.FLOAT MIN 10 EXP

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

Type

INTEGER

Accessibility

Readable From

All subroutines



math.FLOAT MIN EXP

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

Type

INTEGER

Accessibility

Readable From

All subroutines



math.FLOAT_MIN

Minimum finite value for the **FLOAT** type.

Type

FLOAT

Accessibility

Readable From

All subroutines



math.INTEGER BIT

Number of bits in the **INTEGER** type.

Type

INTEGER

Accessibility

Readable From

All subroutines



math.INTEGER MAX

Maximum value for the **INTEGER** type.

Type

INTEGER

Accessibility

Readable From

All subroutines



math.INTEGER MIN

Minimum value for the INTEGER type.

Type

INTEGER

Accessibility

Readable From

All subroutines



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

Type

FLOAT

Accessibility

Readable From

All subroutines



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

Type

FLOAT

Accessibility

Readable From

All subroutines



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

Type

FLOAT

Accessibility

Readable From

All subroutines



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

Type

FLOAT

Accessibility

Readable From

All subroutines



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

Type

FLOAT

Accessibility

Readable From

All subroutines



Negative overflow value.

Type

FLOAT

Accessibility

Readable From

All subroutines



math.NEG INFINITY

A value representing negative infinity ($-\infty$). When converted to a STRING value, this is rendered as $-\inf$.

Type

FLOAT

Accessibility

Readable From

All subroutines



math.PHI

The golden ratio (Φ) .

Type

FLOAT

Accessibility

Readable From

All subroutines



math.Pl 2

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.Pl 4

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.Pl

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.POS HUGE VAL

Positive overflow value.

Type

FLOAT

Accessibility

Readable From

All subroutines



math.POS INFINITY

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.SQRT1 2

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.SQRT2

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

Type

FLOAT

Accessibility

Readable From

All subroutines



math.TAU

The value of <a>math.PI multiplied by two (Tau).

Type

FLOAT

Accessibility

Readable From

All subroutines

Math rounding

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

Math rounding Functions



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

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

Return Value

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

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

If x is math.pos_infinity or math.pos.infinity or <a href="math.pos_infini

Otherwise, the rounded value of x is returned.

Format

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

math.floor()

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

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

Return value

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

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

If x is math.POS INFINITY or math.NEG INFINITY, an infinity of the same sign is returned.

Otherwise, the rounded value of x is returned.

Format

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

math.round()

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

Return value

If x is $\underline{\text{math.NAN}}$, a NaN will be returned.

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

If x is math.pos infinity or infinity</

Otherwise, the rounded value of x is returned.

Format

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

math.roundeven()

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

Return value

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

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

If x is math.pos infinity or math.neg infinity, an infinity of the same sign is returned.

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

Format

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

math.roundhalfdown()

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

Return value

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

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

If x is math.pos_INFINITY or math.neg_INFINITY, an infinity of the same sign is returned.

Otherwise, the rounded value of x is returned.

Format

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

math.roundhalfup()

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

Return value

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

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

If x is math.pos infinity or math.neg infinity, an infinity of the same sign is returned.

Otherwise, the rounded value of x is returned.

Format

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



math.trunc()

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

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

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

Return value

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

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

If x is math.pos_infinity an infinity of the same sign is returned.

Otherwise, the rounded value of x is returned.

Format

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

Math trigonometric

Math trigonometric Functions



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

Parameters

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

Return value

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

If x is $\frac{\text{math.NAN}}{\text{NAN}}$, a NaN will be returned.

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

If x is math.pos infinity or math.neg infinity, a domain error occurs and a NaN will be returned.

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

Errors

If the x argument is finite and is not in the range -1 to 1 inclusive, or is math. POS INFINITY or math. NEG INFINITY, then fastly.e

Format

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

Examples

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

math.acosh()

Computes the inverse hyperbolic cosine of its argument x.

Parameters

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

Return value

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

If x is $\frac{math.NAN}{}$, a NaN will be returned.

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

If x is math.POS INFINITY, math.POS INFINITY will be returned.

If x is $[math.NEG_INFINITY]$, a domain error occurs and a NaN will be returned.

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

Errors

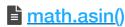
If the x argument is finite and less than +1.0, or is math.neg infinity, then fastly.error will be set to EDOM.

Format

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

Examples

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



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

Parameters

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

Return value

Upon successful completion, this function returns the arc sine of x, in the range - $\frac{math.PI}{2}$ to $\frac{math.PI}{2}$ radians inclusive.

If x is $\frac{math.NAN}{}$, a NaN will be returned.

If x is ± 0 , x will be returned.

If x is math.pos infinity or math.neg infinity, a domain error occurs and a NaN will be returned.

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

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

Errors

- If the x argument is finite and is not in the range -1 to 1 inclusive, or is math.pos Infinity or math.neg Infinity, then fast
- If the x argument is subnormal, then <u>fastly.error</u> will be set to <u>ERANGE</u>.

Format

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

Examples

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

math.asinh()

Computes the inverse hyperbolic sine of its argument x.

Parameters

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

Return value

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

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

If x is ± 0 , or $\frac{\text{math.POS}}{\text{INFINITY}}$ or $\frac{\text{math.NEG}}{\text{INFINITY}}$, x will be returned.

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

Errors

If the x argument is subnormal, then <u>fastly.error</u> will be set to ERANGE.

Format

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

Examples

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

math.atan()

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

Parameters

x - Floating point value.

Return value

Upon successful completion, this function returns the arc tangent of x in the range - math.PI 2 to math.PI 2 radians inclusive.

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

If x is ± 0 , x will be returned.

If x is math.pos infinity or math.neg infinity, ± math.pi 2 will be returned.

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

Errors

If the *x* argument is subnormal, then <u>fastly.error</u> will be set to <u>ERANGE</u>.

Format

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

Examples

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

math.atan2()

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

Parameters

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

Return value

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

If y is ± 0 and x is < 0, $\pm \frac{\text{math.PI}}{\text{will}}$ will be returned.

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

```
If y is < 0 and x is \pm 0, -\frac{math.PI_2}{} will be returned.
```

If y is > 0 and x is ± 0 , $\frac{\text{math.PI}}{2}$ will be returned.

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

If either x or y is $\frac{\text{math.NAN}}{\text{NAN}}$, a NaN will be returned.

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

For finite values of $\pm y > 0$, if x is $\underline{\text{math.NEG_INFINITY}}$, $\pm \underline{\text{math.PI}}$ will be returned.

For finite values of $\pm y > 0$, if x is $\frac{\text{math.POS}}{\text{INFINITY}}$, ± 0 will be returned.

For finite values of x, if y is math.pos_infinity or math.pis_infinity, ±math.pis_infinity will be returned.

If y is $\frac{\text{math.pos infinity}}{\text{math.neg infinity}}$ or $\frac{\text{math.neg infinity}}{\text{math.neg infinity}}$, $\pm (3^* \frac{\text{math.pi 4}}{\text{math.pi 4}})$ will be returned.

If y is math.pos infinity or math.neg infinity and x is math.pos infinity, ± math.pl 4 will be returned.

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

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

Errors

No errors occur.

Format

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

Examples

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

math.atanh()

Computes the inverse hyperbolic tangent of its argument x.

Parameters

x - Floating point value representing a hyperbolic angle.

Return value

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

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

If x is ± 0 , x will be returned.

If x is math.pos_infinity or math.pos.infinity or <a href="math.pos_infini

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

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

If x is ± 1 , a pole error occurs, and $\boxed{\text{math.atanh}()}$ will return the value of the macro $\boxed{\text{math.pos}}$ HUGE $\boxed{\text{VAL}}$ or $\boxed{\text{math.neg}}$ HUGE $\boxed{\text{VAL}}$ will function.

Errors

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

Format

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

Examples

```
declare local var.fo FLOAT;

set var.fo = math.atanh(-1); // Returns math.NEG_INFINITY

if (fastly.error) {
    set resp.http.atanh-error = faslty.error; // Returns "ERANGE"
}
```

math.cos()

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

Parameters

x - Floating point value representing an angle in radians.

Return value

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

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

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

If x is math.pos infinity or math.neg infinity, a domain error occurs and a NaN will be returned.

Errors

• If the x argument is math.pos infinity or math.neg infinity, then fastly.error will be set to EDOM.

Format

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

Examples

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

math.cosh()

Computes the hyperbolic cosine of its argument x.

Parameters

x - Floating point value representing a hyperbolic angle.

Return value

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

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

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

If x is math.pos infinity or math.neg infinity, math.pos infinity will be returned.

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

Errors

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

Format

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

Examples

```
1 declare local var.fo FLOAT;
2
3 set var.fo = math.cosh(0);
```

math.sin()

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

Parameters

x - Floating point value representing an angle in radians.

Return value

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

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

If x is ± 0 , x will be returned.

If x is math.pos infinity or math.neg infinity, a domain error occurs and a NaN will be returned.

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

Errors

- If the x argument is math.pos infinity or math.neg infinity, then fastly.error will be set to EDOM.
- If the x argument is subnormal, then <u>fastly.error</u> will be set to ERANGE.

Format

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

Examples

```
1 declare local var.fi FLOAT;
2 declare local var.fo FLOAT;
3
4 set var.fi = math.PI;
5 set var.fi /= 6;
6 set var.fo = math.sin(var.fi);
```

math.sinh()

Computes the hyperbolic sine of its argument *x*.

Parameters

x - Floating point value representing a hyperbolic angle.

Return value

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

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

If x is ± 0 , or $\underline{\text{math.POS_INFINITY}}$ or $\underline{\text{math.NEG_INFINITY}}$, x will be returned.

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

If the result would cause an overflow, a range error occurs and math.pos Huge val or math.neg Huge val (with the same sign as

Errors

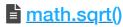
If the x argument is subnormal or if the result would cause an overflow, then fastly.error will be set to ERANGE.

Format

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

Examples

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



Computes the square root of its argument x.

Parameters

x - Floating point value.

Return value

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

If x is $\frac{\text{math.NAN}}{\text{NAN}}$, a NaN will be returned.

If x is ± 0 or $\frac{1}{1}$ math. POS INFINITY, x will be returned.

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

Errors

• If the x argument is < -0 or <pre>math.NEG INFINITY, then fastly.error will be set to EDOM.

Format

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

Examples

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

math.tan()

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

Parameters

x - Floating point value representing an angle in radians.

Return value

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

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

If x is ± 0 , x will be returned.

If x is <u>math.POS_INFINITY</u> or <u>math.NEG_INFINITY</u>, a domain error occurs and a NaN will be returned.

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

If the result would cause an overflow, a range error occurs and <code>math.tan()</code> will return <code>math.POS_HUGE_VAL</code> or <code>math.NEG_HUGE_VAL</code> the function.

Errors

- If the x argument is math.POS_INFINITY or math.NEG_INFINITY, then fastly-error will be set to EDOM.
- If the x argument is subnormal or if the result overflows, then <u>fastly.error</u> will be set to <u>Erange</u>.

Format

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

Examples

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

math.tanh()

Computes the hyperbolic tangent of its argument *x*.

Parameters

x - Floating point value representing a hyperbolic angle.

Return value

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

If x is $\frac{\text{math.NAN}}{\text{nan}}$, a NaN will be returned.

If x is ± 0 , x will be returned.

If x is <u>math.POS INFINITY</u> or <u>math.NEG INFINITY</u>, ±1 will be returned.

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

Errors

If the x argument is subnormal, then <u>fastly.error</u> will be set to <u>ERANGE</u>.

Format

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

Examples

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

Miscellaneous

Miscellaneous features

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

Examples

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

Goto

Similar to some programming languages, goto statements in VCL allow you perform a one-way transfer of control to another line of 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 ac it easier to do things like jump to common error handling blocks before returning from a function. The goto statement works in cust

```
sub vcl_recv {
1
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.

```
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()

Extracts bit_count bits (at most 32) starting with the bit number start_bit from the given IPv4 or IPv6 address and return them in 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). 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 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, 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 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 applied to an address that is neither IPV4 nor IPv6, then it will return 0.

Format

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

Examples

```
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 "07"
3 }
```

addr.is_ipv4()

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

Format

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

Examples

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

addr.is_ipv6()

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

Format

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

Examples

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

http_status_matches()

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

Returns true when the status matches any of the strings and returns false otherwise. If *fmt* is prefixed with 1, returns true when the strings and returns false if it does. Statuses in the string are separated by commas.

This function is not prefixed with the std. namespace.

Format

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

Examples

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



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

You can use <code>if()</code> as a construct to make simple conditional expressions more concise.

Format

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

Examples

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



Returns a value associated with the $\c cookie_name$ in the $\c cet-Cookie$ header contained in the HTTP response indicated by $\c cet$ is not found or on error. In the $\c cet$ method, the $\c cet$ response is available. In $\c cet$ and $\c cet$ and $\c cet$ response is available. In $\c cet$ and $\c cet$ response is available.

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

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

This function conforms to RFC6265.

Format

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

Examples

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



Combines multiple instances of the same header into one. The headers are joined using the optional separator character parameter. 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 si

Format

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

Examples

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

subfield()

Provides a means to access subfields from a header like [Cache-Control], [Cookie], and [Edge-Control] or individual parameters from the control of the contr

The optional separator character parameter defaults to _, . It can be any one-character constant. For example, _; is a useful separate Set-Cookie field.

This functionality is also achievable by using the control: max-age=1200, private), an empty string is returned. The concessor also works for retrieving variables in a cookie.

This function is not prefixed with the std. namespace.

Format

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

Examples

```
if (subfield(beresp.http.Cache-Control, "private")) {
2
     return (pass);
3
   }
4
5
   set beresp.ttl = beresp.http.Cache-Control:max-age;
6
   set beresp.http.Cache-Control:max-age = "1200";
```

```
if (subfield(beresp.http.Set-Cookie, "httponly", ";")) {
2
3
  }
```

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

Miscellaneous Variables



bereq.url.basename

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

Type

STRING

Accessibility

Readable From

All subroutines



bereq.url.dirname

Same as required and your origin servers.

Type

STRING

Accessibility

Readable From

All subroutines



bereg.url.gs

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

Type

STRING

Accessibility

Readable From

All subroutines



bereg.url

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

Type

STRING

Accessibility

Readable From

All subroutines

beresp.backend.ip

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

Type

IP

Accessibility

Readable From

• vcl fetch



beresp.backend.name

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

Type

STRING

Accessibility

Readable From

• vcl fetch



beresp.backend.port

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

Type

INTEGER

Accessibility

Readable From

• vcl fetch



beresp.grace

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

Type

RTIME

Accessibility

Readable From

• vcl_fetch



beresp.hipaa

Specifies that content not be cached in non-volatile memory to help customers meet HIPAA security requirements. See our guide on instructions on enabling this feature for your account.

Type

BOOL

Accessibility

Readable From

• vcl_fetch

beresp.pci

Specifies that content be cached in a manner that satisfies PCI DSS requirements. See our <u>PCI compliance description</u> for instructio account.

Type

BOOL

Accessibility

Readable From

• vcl fetch



The IP address of the client making the request.

Type

IP

Accessibility

Readable From

All subroutines



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 client.port to get a value used both in ESI and the top level request.

Type

INTEGER

Accessibility

Readable From

All subroutines



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

Type

INTEGER

Accessibility

Readable From

All subroutines



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

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

Type

INTEGER

Accessibility

Readable From

All subroutines

fastly.error

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

States

- EPARSENUM: Number parsing failed.
- ERANGE: Numerical result out of range.
- EREGRECUR: Call to regex routine failed because of recursion limits.
- EREGSUB: Call to regex routine failed (generic).
- ESESOOM: Out of workspace memory.
- EDOM: Domain error. This occurs for a mathematical function that is not defined for a particular value. Formally, that value is no domain. For example, division by zero, or [var.x] %= 5; where [var.x] is a floating point infinity.
- ESYNTHOOM: Synthetic response overflow.

Type

STRING

Accessibility

Readable From

All subroutines



reg.backend.healthy

Whether or not this backend, or recursively any of the backends under this director, is considered healthy. The random director has t quorum threshold must be met by the healthy backends under the director. The health state is determined by: healthcheck results, v connection to be made to the backend based on the number of currently used connections and the backend's max connections see saintmode settings.

Type

BOOL

Accessibility

Readable From

- vcl deliver
- vcl_error
- vcl fetch
- vcl hash
- [vcl_hit]
- vcl miss
- vcl pass
- vcl_recv

req.backend.is cluster

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

Type

BOOL

Accessibility

Readable From



req.backend.is origin

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

Type

BOOL

Accessibility

Readable From

- vcl fetch
- vcl miss
- vcl_pass



reg.backend.is shield

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

Type

BOOL

Accessibility

Readable From

All subroutines



req.backend

The backend to use to service the request.

Type

BACKEND

Accessibility

Readable From

All subroutines



reg.body.base64

Same as reg.body, except the request body is encoded in Base64, which handles null characters and allows representation of bina

Type

STRING

Accessibility

Readable From

All subroutines



req.body

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

Type

STRING

Accessibility

Readable From

All subroutines



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

Type

RTIME

Accessibility

Readable From

All subroutines



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

Examples

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

Type

STRING

Accessibility

Readable From

All subroutines



Indicates whether the request was made using IPv6 or not.

Type

BOOL

Accessibility

Readable From

- vcl_recv
- vcl_hash
- vcl_deliver
- vcl_log



Counts the number of times the VCL has been restarted.

Type

INTEGER

Accessibility

Readable From

All subroutines



The file name specified in a URL.

Examples

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

Type

STRING

Accessibility

Readable From

All subroutines



req.url.dirname

The directories specified in a URL.

Examples

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

Type

STRING

Accessibility

Readable From

All subroutines



req.url.ext

The file extension specified in a URL.

Examples

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

Type

STRING

Accessibility

Readable From

All subroutines



req.url.path

The full path, without any query parameters.

Examples

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

Type

STRING

Accessibility

Readable From

All subroutines



req.url.qs

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

Examples

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

Type

STRING

Accessibility

Readable From

All subroutines



The full path, including query parameters.

Examples

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

Type

STRING

Accessibility

Readable From

All subroutines



stale.exists

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

Type

STRING

Accessibility

Readable From

All subroutines

Query string manipulation

Examples

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

```
sub vcl_recv {
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

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

Query string manipulation Functions



boltsort.sort()

Alias of guerystring.sort.

Format

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

Examples

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

auerystring.add()

Returns the given URL with the given parameter name and value appended to the end of the query string. The parameter name and added to the query string.

Format

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

Examples

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



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

Format

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

Examples

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

auerystring.filter_except()

Returns the given URL but only keeps the listed parameters.

Format

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

Examples

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

auerystring.filter()

Returns the given URL without the listed parameters.

Format

```
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");
```

auerystring.filtersep()

Returns the separator needed by the guerystring.filter except() functions.

Format

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

Examples

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

a <u>querystring.globfilter</u> except()

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

Format

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

Examples

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



auerystring.globfilter()

Returns the given URL without the parameters matching a glob.

Format

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

Examples

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



<u>auerystring.regfilter</u> <u>except()</u>

Returns the given URL but only keeps the parameters matching a regular expression. Groups within the regular expression are treate capturing groups. For example:

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

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

Format

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

Examples

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



auerystring.regfilter()

Returns the given URL without the parameters matching a regular expression. Groups within the regular expression are treated as if groups. For example:

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

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

Format

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

Examples

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



auerystring.remove()

Returns the given URL with its query-string removed.

Format

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

Examples

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



auerystring.set()

Returns the given URL with the given parameter name set to the given value, replacing the original value and removing any duplicate the query string, the parameter will be appended with the given value to the end of the query string. The parameter name and value v the query string.

Format

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

Examples

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



auerystring.sort()

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

Format

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

Examples

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

Randomness

▲ WARNING: We use BSD random number functions from the GNU C Library, not true randomizing sources. These VCL function 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-

```
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

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

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

Random decisions

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

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

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

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

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

Another function, randombool_seeded(), takes an additional seed argument. Results for a given seed will always be the same. For i 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 cookie.

▲ WARNING: The randombool and randombool_seeded functions do not use secure random numbers and should not be used

Randomness Functions

andombool_seeded()

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

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

This function is not prefixed with the std. namespace.

Format

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

Examples

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

randombool()

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

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

This function is not prefixed with the std. namespace.

Format

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

Examples

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

randomint seeded()

Identical to randomint, except takes an additional parameter used to seed the random number generator.

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

This function is not prefixed with the std. namespace.

Format

```
INTEGER
randomint_seeded(INTEGER from, INTEGER to, INTEGER seed)
```

Examples

```
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 }</pre>
```

randomint()

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

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

This function is not prefixed with the std. namespace.

Format

```
INTEGER randomint(INTEGER from, INTEGER to)
```

Examples

```
1  if (randomint(0, 99) < 5) {
2    set req.http.X-ABTest = "A";
3  } else {
4    set req.http.X-ABTest = "B";
5  }
6  if (randomint(-1, 0) == -1) {
7    set req.http.X-ABTest = "A";
8  } else {
9    set req.http.X-ABTest = "B";
10 }</pre>
```

in randomstr()

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

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

This function is not prefixed with the std. namespace.

Format

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

Examples

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

Segmented Caching

Segmented Caching Variables



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

Type

BOOL

Accessibility

Readable From

• vcl log

fastly.segmented_caching.block_number

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

Type

INTEGER

Accessibility

Readable From

• vcl log



fastly.segmented caching.cancelled

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

Type

BOOL

Accessibility

Readable From

• vcl log



fastly.segmented caching.client reg.is open ended

Whether the client's request leaves the upper bound of the range open. This variable will evaluate to false when Segmented Cachi

Type

BOOL

Accessibility

Readable From

• vcl_log



fastly.segmented caching.client req.is range

Whether the client's request is a range request. This variable with evaluate to false when Segmented Caching is not enabled for th is present).

Type

BOOL

Accessibility

Readable From

• vcl log



fastly.segmented caching.client reg.range high

The upper bound of the client's requested range. This variable will evaluate to [-1] in cases when it is not applicable, such as when S for the request. It will evaluate to 9223372036854775807 (2^63-1) for an open-ended requested range (when fastly.segmented caching.client reg.is open ended is true).

Type

INTEGER

Accessibility

Readable From

• vcl log



fastly.segmented caching.client req.range low

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

Type

INTEGER

Accessibility

Readable From

• vcl_log



fastly.segmented_caching.completed

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

Type

BOOL

Accessibility

Readable From

• vcl log



fastly.segmented caching.error

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

Type

STRING

Accessibility

Readable From

• vcl log



fastly.segmented caching.failed

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

Type

BOOL

Accessibility

Readable From

• vcl_log



fastly.segmented caching.is inner req

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

Type

BOOL

Accessibility

Readable From

• vcl log



fastly.segmented caching.is outer req

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

Type

BOOL

Accessibility

Readable From

• vcl_log



fastly.segmented_caching.obj.complete_length

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

Type

INTEGER

Accessibility

Readable From

• vcl log



fastly.segmented caching.rounded req.range high

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

Type

INTEGER

Accessibility

Readable From

• vcl log



fastly.segmented caching.rounded req.range low

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

Type

INTEGER

Accessibility

Readable From

• vcl log



fastly.segmented caching.total blocks

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

Type

INTEGER

Accessibility

Readable From

• vcl_log

Server

Server Variables



server.datacenter

A code representing one of Fastly's POP locations.

Type

STRING

Accessibility

Readable From

All subroutines



server.hostname

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

Type

STRING

Accessibility

Readable From

All subroutines



server.identity

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

Type

STRING

Accessibility

Readable From

All subroutines



server.region

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

Region Name	Approximate Geographic Location of Fastly POPs
APAC	Australia and New Zealand
Asia	throughout the Asian continent (except India)
Asia-South	southern Asia
EU-Central	the central European continent
EU-East	the eastern European continent
EU-West	the western European continent
North-America	Canada
SA-East	eastern South America
SA-North	northern South America
SA-South	southern South America
South-Africa	the southern regions of Africa
US-Central	the central United States
US-East	the eastern United States
US-West	the western United States

Type

STRING

Accessibility

Readable From

All subroutines

Size

Size Variables



bereq.body bytes written

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

Type

INTEGER

Accessibility

Readable From

- vcl_fetch
- vcl deliver
- vcl_log



bereq.header_bytes_written

Total header bytes written to a backend.

Type

INTEGER

Accessibility

Readable From

- vcl fetch
- vcl_deliver
- vcl_log



req.body bytes read

Total body bytes read from the client generating the request.

Type

STRING

Accessibility

Readable From

- vcl_deliver
- vcl_log



req.bytes_read

Total bytes read from the client generating the request.

Type

STRING

Accessibility

Readable From

- vcl deliver
- vcl_log



req.header bytes read

Total header bytes read from the client generating the request.

Type

STRING

Accessibility

Readable From

All subroutines



resp.body bytes written

Body bytes to send to the client in the response.

Type

STRING

Accessibility

Readable From

• vcl log



resp.bytes written

Total bytes to send to the client in the response.

Type

STRING

Accessibility

Readable From

• vcl log



resp.completed

Whether the response completed successfully or not.

Type

BOOL

Accessibility

Readable From

• vcl_log



resp.header bytes written

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

Type

STRING

Accessibility

Readable From

• vcl_log

String manipulation

String manipulation Functions



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

The escaping rules in priority order are as follows:

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

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

This function is not prefixed with the std. namespace.

Format

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

Examples

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

json.escape()

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

Format

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

Examples

```
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"}
```



Replaces the first occurrence of pattern, which is a Perl-compatible regular expression, in input with replacement. If no match 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 lookaher other recursing non-regular expressions. In this case, <u>fastly.error</u> is set to <u>EREGRECUR</u>.

This function is not prefixed with the std. namespace.

Format

```
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()

Replaces all occurrences of pattern, which may be a Perl-compatible regular expression, in input with replacement. If no match made.

Once a replacement is made, substitutions continue from the end of the replaced buffer. Therefore, regsuball("aaa", "a", "aa")
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 lookaher other recursing non-regular expressions. In this case, <u>fastly.error</u> is set to <u>EREGRECUR</u>.

This function is not prefixed with the std. namespace.

Format

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

Examples

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

std.anystr2ip()

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

This function accepts a wider range of formats than std.str2ip(): Each number may be specified in hexadecimal (0x...), octal (may be fewer than four numbers, in which case the last number is responsible for the remaining bytes of the IP. For example, 0x8.01

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

Format

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

Examples

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

std.atof()

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

Format

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

Examples

```
1 if (std.atof(req.http.X-String) > 21.82) {
2  set req.http.X-TheAnswer = "Found";
3 }
```

std.atoi()

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

Format

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

Examples

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

std.ip()

An alias of std.str2ip().

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

Format

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

Examples

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

std.ip2str()

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

Format

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

Examples

```
1 if (std.ip2str(std.str2ip(req.http.Fastly-Client-IP, "192.0.2.2")) ~ my_acl) {
2   ...
3 }
```

std.prefixof()

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

Returns false otherwise.

Format

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

Examples

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



Converts the string representation of an IP address (IPv4 or IPv6) into an <u>IP type</u>. If conversion fails, the fallback will be returned. T address representation in the standard format such as <u>192.0.2.2</u> and <u>2001:db8::1</u>. This function does not support looking up an

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

Format

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

Examples

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

std.strlen()

Returns the length of the string. For example, [std.strlen("Hello world!"); will return 12 (because the string includes whitespace)

Format

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

Examples

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

std.strpad()

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

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

Negative width left-justifies s by placing padding to the right. Positive width right-justifies s by placing padding to the left. If will length of s, then no padding is performed.

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

Format

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

Examples

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

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

std.strrep()

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

Format

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

Examples

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



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

Errors

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

Format

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

Examples

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

std.strstr()

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

Format

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

Examples

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

std.strtof()

Converts the string s to a float value with the given base base. The value base must be a constant integer expression (variables are r

The following string formats are supported for finite values:

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

The syntax for these values corresponds to the syntax for VCL FLOAT literals in base 10 and 16 respectively. See <u>VCL Types</u> for deta Supported bases are 0, 10, or 16.

A base of 0 causes the base to be automatically determined from the string format. In this case, a ①x prefix indicates hex (base 16), decimal (base 10).

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

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

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

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

The NaN and infinity special formats are case sensitive.

No whitespace is permitted by std.strtof.

On error, <u>fastly.error</u> is set.

Format

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

Examples

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

std.strtol()

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

The following string formats are supported:

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

The syntax for integers extends the syntax for VCL INTEGER literals in base 10 and 16 respectively. See <u>VCL Types</u> for details of the bases.

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

A base of 0 causes the base to be automatically determined from the string format. In this case, a ①x prefix indicates hex (base 16), 8), and otherwise the base is taken as decimal (base 10).

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

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

If the base is outside the range, or the number exceeds the range of a signed integer, <u>fastly.error</u> is set to <u>ERANGE</u>. If the numbe <u>fastly.error</u> is set to <u>EPARSENUM</u>.

On error, <u>fastly.error</u> is set.

Format

```
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()

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

Returns false otherwise.

Format

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

Examples

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



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

Format

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

Examples

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

std.toupper()

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

Format

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

Examples

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

substr()

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

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

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

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

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

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

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

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

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

Format

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

Examples

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

urldecode()

Decodes a percent-encoded string. For example, <code>urldecode({"hello%20world+!"});</code> and <code>urldecode("hello%2520world+!");</code> w

Format

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

Examples

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



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

Format

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

Examples

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

utf8.codepoint count()

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

Format

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

utf8.is_valid()

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

Format

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

utf8.strpad()

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

Errors

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

Format

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

Examples

utf8.substr()

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

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

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

1 NOTE: UTF-8 allows you to combine characters, which are separate code points. While <code>utf8.substr()</code> correctly honors the blowever, requesting a substring of several of them may not necessarily represent a meaningful grapheme cluster.

Format

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

Table

Tables are declared as follows:

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

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

Table Functions

table.lookup()

Look up the key key in the table lb. When the key is present, its associated value will be returned. When the key is absent, the value will be returned. When the key is absent, the value will be returned. When a key is absent, the defa

Format

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

Examples

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

```
1
  table geoip_lang {
     "US": "en-US",
2
     "FR": "fr-FR",
3
     "NL": "nl-NL",
4
5
  }
6
  if (!req.http.Accept-Language) {
7
     set req.http.Accept-Language = table.lookup(geoip_lang, geoip.country_code, "en-US");
8
  }
```

TLS and HTTP/2

When using these variables, remember the following:

- These variables are currently only allowed to appear within the VCL hooks vcl recv, vcl hash, vcl deliver and vcl log
- Requests made with HTTP/2 will appear in custom logs as HTTP/1.1 because those requests will already have been decrypted Specifically, the Fr variable will not accurately represent the type of HTTPX request being processed.

TLS and HTTP/2 Functions



h2.disable header compression()

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

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

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

Format

```
h2.disable header compression(STRING header)
```

Examples

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



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

Format

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

Examples

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

TLS and HTTP/2 Variables



fastly info.h2.is push

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

Type

BOOL

Accessibility

Readable From

- vcl recv
- vcl hash
- vcl deliver
- vcl_log

fastly info.h2.stream id

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

Type

INTEGER

Accessibility

Readable From

- vcl recv
- vcl_hash
- vcl deliver
- vcl_log

fastly info.is h2

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

Type

BOOL

Accessibility

Readable From

- vcl_recv
- vcl_hash
- vcl_deliver
- vcl_log

tls.client.cipher

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

Examples

"ECDHE-RSA-AES128-GCM-SHA256"

Type

STRING

Accessibility

Readable From

- vcl_recv
- vcl hash
- vcl_deliver
- [vcl_log]

tls.client.ciphers list sha

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

Type

STRING

Accessibility

Readable From

- vcl_recv
- vcl_hash
- vcl_deliver
- vcl log



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

Type

STRING

Accessibility

Readable From

- vcl recv
- vcl_hash
- vcl_deliver
- vcl_log

tls.client.ciphers list

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

Type

STRING

Accessibility

Readable From

- [vcl_recv]
- vcl_hash
- vcl deliver
- vcl_log

tls.client.ciphers_sha

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

Type

STRING

Accessibility

Readable From

- vcl_recv
- vcl hash
- vcl_deliver
- vcl_log

tls.client.protocol

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

Type

STRING

Accessibility

Readable From

- vcl_recv
- vcl_hash
- vcl_deliver
- vcl_log

tls.client.servername

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

Type

STRING

Accessibility

Readable From

- vcl_recv
- vcl_hash
- vcl deliver
- vcl_log

tls.client.tlsexts list sha

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

Type

STRING

Accessibility

Readable From

- vcl_recv
- vcl_hash
- vcl_deliver
- vcl log

tls.client.tlsexts list txt

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

Type

STRING

Accessibility

Readable From

- [vcl_recv]
- vcl_hash
- vcl deliver

• vcl log

tls.client.tlsexts list

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

Type

STRING

Accessibility

Readable From

- vcl recv
- vcl_hash
- vcl_deliver
- vcl log

tls.client.tlsexts_sha

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

Type

STRING

Accessibility

Readable From

- vcl_recv
- vcl_hash
- vcl_deliver
- vcl log

UUID

UUID Functions



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

Format

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

Examples

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

uuid.is valid()

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

Format

```
BOOL
uuid.is_valid(STRING string)
```

Examples

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

uuid.is_version3()

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

Format

```
BOOL uuid.is_version3(STRING string)
```

Examples

```
1 if (uuid.is_version3(req.http.X-Unique-Id)) {
2   set beresp.http.X-Unique-Id-Valid-V3 = "yes";
3 }
```

uuid.is_version4()

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

Format

```
BOOL uuid.is_version4(STRING string)
```

Examples

```
1 if (uuid.is_version4(req.http.X-Unique-Id)) {
2   set beresp.http.X-Unique-Id-Valid-V4 = "yes";
3 }
```

uuid.is_version5()

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

Format

```
BOOL
uuid.is_version5(STRING string)
```

Examples

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

uuid.oid()

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

Format

```
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 "31cblefa-18c4-3d19-89ba-df6a74ddbd1d"
```



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

Format

```
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()

Derives a UUID corresponding to name within the given namespace using MD5 hash function. Namespace itself is identified by a UI form appropriate for selected namespace.

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

Format

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

Examples

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



Returns a UUID based on random number generator output.

Format

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

Examples

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



Derives a UUID corresponding to name within the given namespace using SHA-1 hash function. Namespace itself is identified by a I form appropriate for selected namespace.

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

Format

```
STRING
uuid.version5(STRING namespace, STRING name)
```

Examples

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



Returns the RFC4122 identifier of X.500 namespace, namely the constant ["6ba7b812-9dad-11d1-80b4-00c04fd430c8"].

Format

```
STRING
uuid.x500()
```

Examples

```
1 declare local var.x500 STRING;
2 set var.x500 = uuid.version3(uuid.x500(), "CN=Test User 1, 0=Example Organization, ST=California, C=US");
3 # var.x500 is now "addf5e97-9287-3834-abfd-7edcbe7db56f"
Guides
```

§ Custom VCL

Creating custom VCL

Fastly Varnish syntax is specifically compatible with <u>Varnish 2.1.5</u>. We run a custom version with added functionality and our VCL pa 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 <code>curl -X PURGE</code> to initiate purges via API. To restrict access to purging, check for the <code>FASTLYPURGE</code> method send a request to Varnish to initiate a purge, the HTTP method that you use is "PURGE", but it has already been changed to "F VCL runs that request.
- If you override TTLs with custom VCL, your default TTL set in the configuration will not be honored and the expected be

1 IMPORTANT: Personal data should not be incorporated into VCL. Our Compliance and Law FAQ describes in detail how Fastly

Inserting custom VCL in Fastly's VCL boilerplate

▲ DANGER: Start with the VCL boilerplate and add your custom code *in between* the different sections as shown in the example override the VCL at that point. You must include all of the Fastly VCL boilerplate as a template in your custom VCL file, especially with #FASTLY). VCL macros expand the code into generated VCL.

Custom VCL placement example

```
1  sub vcl_miss {
2    # my custom code
3    if (req.http.User-Agent ~ "Googlebot") {
4       set req.backend = F_special_google_backend;
5    }
6  #FASTLY miss
7    return(fetch);
8 }
```

Fastly's VCL boilerplate

★ TIP: If you use the Fastly Image Optimizer, use the image optimization VCL boilerplate instead.

```
sub vcl_recv {
 1
 2
    #FASTLY recv
 3
      if (req.method != "HEAD" && req.method != "GET" && req.method != "FASTLYPURGE") {
 4
 5
        return(pass);
 6
 7
 8
      return(lookup);
 9
    }
10
    sub vcl fetch {
11
12
    #FASTLY fetch
13
14
      if ((beresp.status == 500 || beresp.status == 503) && req.restarts < 1 && (req.method == "GET" || req.method == "
15
16
17
18
      if (req.restarts > 0) {
19
        set beresp.http.Fastly-Restarts = req.restarts;
20
21
22
      if (beresp.http.Set-Cookie) {
        set req.http.Fastly-Cachetype = "SETCOOKIE";
23
24
        return(pass);
25
      }
26
27
      if (beresp.http.Cache=Control ~ "private") {
28
        set req.http.Fastly-Cachetype = "PRIVATE";
29
        return(pass);
30
31
32
      if (beresp.status == 500 || beresp.status == 503) {
        set req.http.Fastly-Cachetype = "ERROR";
33
34
        set beresp.ttl = 1s;
35
        set beresp.grace = 5s;
36
        return(deliver);
37
38
39
      if (beresp.http.Expires || beresp.http.Surrogate-Control ~ "max-age" || beresp.http.Cache-Control ~ "(s-maxage|ma
40
        # keep the ttl here
41
      } else {
42
        # apply the default ttl
43
        set beresp.ttl = 3600s;
44
45
      return(deliver);
46
    }
47
48
49
    sub vcl_hit {
50
    #FASTLY hit
51
      if (!obj.cacheable) {
52
53
        return(pass);
      }
54
55
      return(deliver);
    }
56
57
58
    sub vcl miss {
59
    #FASTLY miss
      return(fetch);
60
61
    }
62
63
    sub vcl_deliver {
64
    #FASTLY deliver
65
      return(deliver);
66
    }
67
68
    sub vcl_error {
69
    #FASTLY error
```

```
70  }
71
72  sub vcl_pass {
73  #FASTLY pass
74  }
75
76  sub vcl_log {
77  #FASTLY log
78  }
```

Uploading custom VCL

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

IMPORTANT: Personal data should not be incorporated into VCL. Our <u>Compliance and Law FAQ</u> describes in detail how Fastly

Uploading a VCL file

Follow these instructions to upload a custom VCL file:

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

Our VCL tutorial will help you get started with creating VCL files. Name My custom VCL For included files, this name must exactly match the include statement in the main VCL file. Config file UPLOAD FILE custom.vcl

Upload a new VCL file

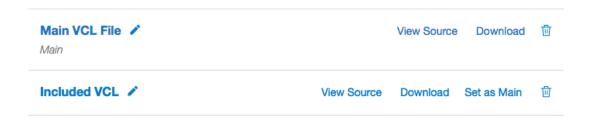
6. In the **Name** field, enter the name of the VCL file. For included files, this name must match the include statement in the main VC additional VCL configurations for more information.

CANCEL

7. Click **Upload file** and select a file to upload. The name of the uploaded file appears next to the button.

CREATE

- **1 IMPORTANT:** Don't upload generated VCL that you've downloaded from the Fastly web interface. Instead, edit and then boilerplate to avoid errors.
- 8. Click the Create button. The VCL file appears in the Varnish Configurations area.

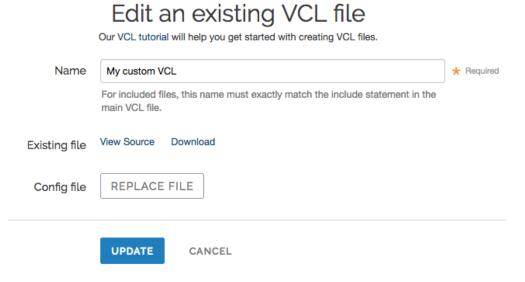


9. Click the Activate button to deploy your configuration changes.

Editing a VCL file

To edit an existing VCL file, follow these instructions:

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



- 6. In the Name field, optionally enter a new name of the VCL file.
- 7. Click the **Download** link to download the appropriate file.
- 8. Make the necessary changes to your file and save them.
- 9. Click the Replace file button and select the file you updated. The selected file replaces the current VCL file and the file name a
- 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 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 <u>access control list</u> for code manageability) and the file VCL configuration file would need to contain this line:

Previewing and testing VCL

Any time you upload VCL files you can preview and test the VCL prior to activating a new version of your service.

Previewing VCL before activation

To preview VCL prior to activating a service version.

- 1. Log in to the Fastly web interface and click the Configure link.
- 2. From the service menu, select the appropriate service.
- 3. Click the Edit configuration button and then select Clone active. The Domains page appears.
- 4. Click the Show VCL link.





The VCL preview page appears.

Testing VCL configurations

You don't need a second account to test your VCL configurations. We recommend adding a new service within your existing accoun 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 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 tile to the production service you'll be using.

§ VCL Snippets

About VCL Snippets

VCL Snippets are short blocks of <u>VCL logic</u> that can be included directly in your service configurations. They're ideal for adding smalneed more complex, specialized configurations that sometimes require <u>custom VCL</u>. Fastly supports two types of VCL Snippets:

- Regular VCL Snippets get created as you create versions of your Fastly configurations. They belong to a specific service and snippet are locked and deployed when you deploy a new version of that service. You can treat regular snippets like any other F to clone them and deploy them with a service until you specifically delete them. You can create regular snippets using either th
- <u>Dynamic VCL Snippets</u> can be modified and deployed any time they're changed. Because they are versionless objects (much the edge), dynamic snippets can be modified independently from service changes. This means you can modify snippet code raversion that may not be ready for production. You can only create dynamic snippets via the API.

Limitations of VCL Snippets

- Snippets are limited to 1MB in size by default. If you need to store snippets larger than the limit, contact support@fastly.com.
- Snippets don't currently support conditions created through the web interface. You can, however, use if statements in snippe
- Snippets cannot currently be shared between services.

Using dynamic VCL Snippets

Dynamic VCL Snippets are one of <u>two types of snippets</u> that allow you to insert small sections of VCL logic into your service configu <u>VCL</u> (though you can still <u>include snippets in custom VCL</u> when necessary).

You can only create dynamic snippets via the API. Because they are versionless objects (much like <u>Edge Dictionaries</u> or <u>ACLs</u> at the modified independently from changes to your Fastly service. This means you can modify snippet code rapidly without deploying a set 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_
plication/x-www-form-urlencoded' --data $'name=my_dynamic_snippet_name&type=recv&dynamic=1&content=if ( req.url ) {\n s
der = "true";\n}';
```

Fastly returns a JSON response that looks like this:

```
1
    {
     "service_id": "<Service Id>",
 2
     "version": "<Editable Version>"
 3
 4
     "name": "my_dynamic_snippet_name",
     "type": "recv",
 5
     "priority": 100,
 6
     "dynamic": 1,
 8
     "content": null,
 9
     "id": "decafbad12345",
     "created_at": "2016-09-09T20:34:51+00:00",
10
     "updated_at": "2016-09-09T20:34:51+00:00",
11
12
     "deleted_at": null
13 }
```

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

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 slocation in

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

Dynamic snippets

These are the dynamic snippets currently in use. You can only edit them via the API because they are not versioned.

My snippet that is dynamic	View source	Show in generated VCI
Priority: 10		
Type: init		
My other snippet that is dynamic	View source	Show in generated VCI
Priority: 10		
Type: init		
My third snippet that is dynamic	View source	Show in generated VCI
Priority: 10		
Type: init		

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 A

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_TO
```

Unlike fetching regular VCL Snippets, you do not include the version in the URL and you must use the ID returned when the snippet

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_TO ion/x-www-form-urlencoded' --data $'content=if(req.url) {\n set req.http.my-snippet-test-header = \"affirmative\"; \n set req.http.my-snippet-test-header = \"affirmative\"; \n$

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_
_TOKEN"</pre>

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 it 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 all that looks at all incoming requests and generates a set of rules that can identify scrapers using a combination of the incoming IP add 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 latest version of the service since it might be untested. Instead you could generate the rules as a Dynamic VCL Snippet. Whenev logic remains the same as the currently deployed version and only your rules are modified.

Using regular VCL Snippets

Regular VCL Snippets are one of two types of snippets that allow you to insert small sections of VCL logic into your service configuration (though you can still include snippets in custom VCL when necessary).

Unlike <u>dynamic snippets</u>, regular snippets can be created via the web interface or via the API. They are considered "versioned" object service and any modifications you make to the snippet are locked and deployed when you deploy a new version of that service. We 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.

<u> </u>		L _	_	\ /	1
(r	<i>'ea</i>	\Box	$\boldsymbol{\mathcal{A}}$	\ /	(
\bigcirc	Ca		a	V	•

VCL snipp

Name		
------	--	--

Type (placement of the snippet)

This specifies the location in which

- init inserts the snippets above
 access control lists, tables)
- within subroutine inserts the boilerplate code and preceeding recv (vcl_recv)
 - O none (advanced) requires you VCL

VCL 1



- 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 sel
 - Select none (advanced) to insert it manually. See <u>Including regular snippets in custom VCL</u> for the additional manual inser 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_
H 'Content-Type: application/x-www-form-urlencoded' --data $'name=my_regular_snippet&type=recv&dynamic=0&content=if ( r
ippet-test-header = "true";\n}';
```

Fastly returns a JSON response that looks like this:

```
1
     "service_id": "<Service Id>",
 2
     "version": "<Editable Version>",
 3
     "name": "my_regular_snippet",
 4
 5
     "type": "recv",
     "content": "if ( req.url ) {\n set req.http.my-snippet-test-header = \"true\";\n}",
 6
     "priority": 100,
 7
     "dynamic": 0,
 8
     "id": "56789exampleid",
 9
     "created_at": "2016-09-09T20:34:51+00:00",
10
     "updated_at": "2016-09-09T20:34:51+00:00",
11
     "deleted_at": null
12
13 }
```

3 NOTE: When regular VCL snippets get created, an <u>id</u> field will be returned that isn't used. The field only applies to <u>dynamic V</u> returned JSON includes a populated <u>content</u> field because the snippet content is stored in a versioned object.

Viewing regular VCL Snippets in the web interface

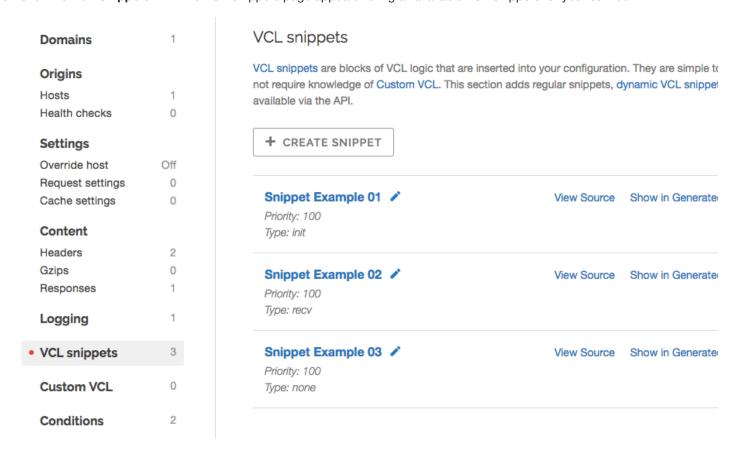
You can view a list of regular VCL snippets. You can also view just the source of a specific snippet or a specific snippet 's location in

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_re ASTLY_API_TOKEN" Unlike fetching dynamic VCL Snippets you include the version in the URL and you must use the name of the snippet, not the ID.

Fetching a list of regular VCL Snippets

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

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

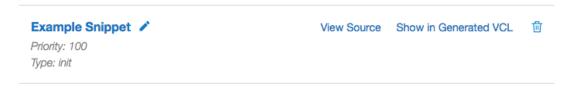
Updating an existing regular VCL Snippet

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.

Edit snippet

VCL snippet guide

Name	Example Snippet	* Required
Type (placement of the snippet)	This specifies the location in which to place the snippet init - inserts the snippets above all subroutines (good for defining backends, access control lists, tables) within subroutine - inserts the snippets within a subroutine (following any boilerplate code and preceeding any objects) none (advanced) - requires you to manually insert the snippet using custom VCL	
VCL	Example Snippet VCL	
> Advanced option	Priority	
	UPDATE CANCEL	

- 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:

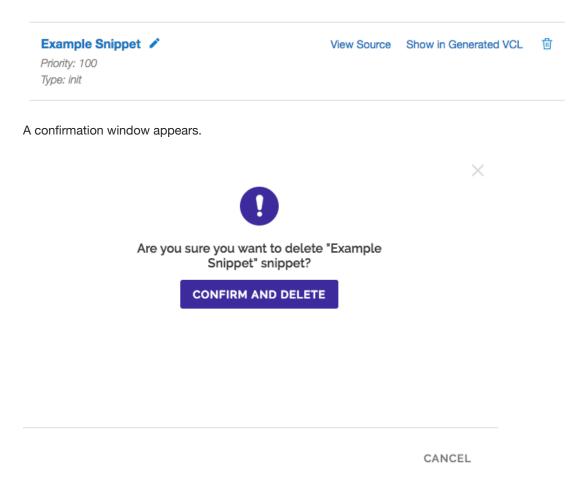
curl -X PUT -s https://api.fastly.com/service/<Service ID>/version/<Editable Version #>/snippet/<Snippet Name e.g my_re
ASTLY_API_TOKEN" -H 'Content-Type: application/x-www-form-urlencoded' --data \$'content=if (req.url) {\n set req.http.rmative\";\n}';</pre>

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.



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
y:FASTLY_API_TOKEN"</pre>

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 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 v 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" || ) {
2
     set req.http.Edition = "US";
3
      set req.backend = F_US;
4 } elseif (reg.http.Cookie:edition == "Europe" || server.region ~ "^EU-" ) {
5
     set req.http.Edition = "EU";
6
    set req.backend = F_European;
7 } else {
      set req.http.Edition = "INT";
8
9
      set req.backend = F_International;
10 }
```

This would create an Edition header in VCL, but allow you to override it by setting a condition. You would <u>add the Edition header into condition</u> (e.g., <u>!reg.url</u>) to your other backends to ensure the correct edition of your publication gets delivered (Remember: VCL stackends are set.)

§ VCL Reference



These VCL functions are supported by Fastly.

Content negotiation

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

- <u>accept.charset_lookup()</u> Selects the best match from a string in the format of an <u>Accept-Charset</u> header's value in the liste algorithm described in Section 5.3.3 of RFC 7231.
- <u>accept.encoding lookup()</u> Selects the best match from a string in the format of an <u>Accept-Encoding</u> header's value in the I algorithm described in Section 5.3.3 of RFC 7231.
- <u>accept.language_filter_basic()</u> Similar to <u>accept.language_lookup()</u>, this function selects the best matches from a string in <u>Language</u> header's value in the listed languages, using the algorithm described in RFC 4647, Section 3.3.1.
- <u>accept.language lookup()</u> Selects the best match from a string in the format of an <u>Accept-Language</u> header's value in the I algorithm described in RFC 4647, Section 3.4.
- accept.media lookup() Selects the best match from a string in the format of an Accept header's value in the listed media ty
 in Section 5.3.2 of RFC 7231.

Cryptographic

Fastly provides several functions in <u>VCL</u> for cryptographic- and hashing-related purposes. It is based very heavily on Kristian Lyngste (which means you can also refer to that documentation for more detail).

- digest.awsv4 hmac() Returns an AWSv4 message authentication code based on the supplied key and string.
- <u>digest.base64_decode()</u> Returns the Base64 decoding of the input string, as specified by RFC 4648.
- <u>digest.base64()</u> Returns the Base64 encoding of the input string, as specified by RFC 4648.
- <u>digest.base64url_decode()</u> Returns the Base64 decoding with URL and filename safe alphabet decoding of the input string,
- <u>digest.base64url_nopad_decode()</u> Returns the Base64 decoding with URL and filename safe alphabet decoding of the input without padding (=).

- digest.base64url_nopad() Returns the Base64 encoding with URL and filename safe alphabet encoding of the input string, a padding (=).
- <u>digest.base64url()</u> Returns the Base64 encoding with URL and filename safe alphabet of the input string, as specified by RF
- <u>digest.hash_crc32()</u> Calculates the 32-bit Cyclic Redundancy Checksum with reversed bit ordering of a string, like that used
- <u>digest.hash_crc32b()</u> Calculates the 32-bit Cyclic Redundancy Checksum of a string, as specified by ISO/IEC 13239:2002 a recommendation V.42 and used by Ethernet (IEEE 802.3), V.42, FDDI, gzip, zip, and PNG.
- <u>digest.hash md5()</u> Use the MD5 hash.
- digest.hash sha1() Use the SHA-1 hash.
- digest.hash sha224() Use the SHA-224 hash.
- digest.hash sha256() Use the SHA-256 hash.
- digest.hash sha384() Use the SHA-384 hash.
- digest.hash sha512() Use the SHA-512 hash.
- digest.hmac md5 base64() Hash-based message authentication code using MD5.
- <u>digest.hmac_md5()</u> Hash-based message authentication code using MD5.
- <u>digest.hmac_sha1_base64()</u> Hash-based message authentication code using SHA-1.
- <u>digest.hmac sha1()</u> Hash-based message authentication code using SHA-1.
- digest.hmac sha256 base64() Hash-based message authentication code using SHA-256.
- <u>digest.hmac sha256()</u> Hash-based message authentication code using SHA-256.
- <u>digest.hmac sha512 base64()</u> Hash-based message authentication code using SHA-512.
- <u>digest.hmac sha512()</u> Hash-based message authentication code using SHA-512.
- <u>digest.rsa_verify()</u> A boolean function that returns true if the RSA signature of payload using public_key matches digest
- <u>digest.secure is equal()</u> A boolean function that returns true if s1 and s2 are equal.
- <u>digest.time_hmac_md5()</u> Returns a time-based one-time password using MD5 based upon the current time.
- <u>digest.time_hmac_sha1()</u> Returns a time-based one-time password using SHA-1 based upon the current time.
- digest.time hmac sha256() Returns a time-based one-time password with SHA-256 based upon the current time.
- digest.time hmac sha512() Returns a time-based one-time password with SHA-512 based upon the current time.

Date and time

By default VCL includes the now variable, which provides the current time (for example, Mon, 02 Jan 2006 22:04:05 GMT). Fastly and functions that allow more flexibility when dealing with dates and times.

- parse time delta() Parses a string representing a time delta and returns an integer number of seconds.
- std.integer2time() Converts an integer, representing seconds since the UNIX Epoch, to a time variable.
- <u>std.time()</u> Converts a string to a time variable.
- <u>strftime()</u> Formats a time to a string.
- time.add() Adds a relative time to a time.
- time.hex to time() This specialized function takes a hexadecimal string value, divides by divisor and interprets the result a
- <u>time.is_after()</u> Returns true if [t1] is after [t2].
- time.sub() Subtracts a relative time from a time.

Floating point classification

Floating point classification functions.

- math.is finite() Determines whether a floating point value is finite.
- math.is_infinite() Determines whether a floating point value is an infinity.
- math.is nan() Determines whether a floating point value is NaN (Not a Number).
- math.is normal() Determines whether a floating point value is normal.
- math.is subnormal() Determines whether a floating point value is subnormal.

Math rounding

Rounding of numbers.

- math.ceil() Computes the smallest integer value greater than or equal to the given value.
- math.floor() Computes the largest integer value less than or equal to the given value.
- math.round() Rounds x to the nearest integer, with ties away from zero (commercial rounding).
- <u>math.roundeven()</u> Rounds *x* to nearest, ties to even (bankers' rounding).
- math.roundhalfdown() Rounds to nearest, ties towards negative infinity (half down).
- <u>math.roundhalfup()</u> Rounds to nearest, ties towards positive infinity (*half up*).
- math.trunc() Truncates x to an integer value less than or equal in absolute value.

Math trigonometric

Trigonometric functions.

- math.acos() Computes the principal value of the arc cosine of its argument x.
- math.acosh() Computes the inverse hyperbolic cosine of its argument x.
- math.asin() Computes the principal value of the arc sine of the argument x.
- math.asinh() Computes the inverse hyperbolic sine of its argument x.
- math.atan() Computes the principal value of the arc tangent of its argument x.
- math.atan2() Computes the principal value of the arc tangent of y/x, using the signs of both arguments to determine the gua
- math.atanh() Computes the inverse hyperbolic tangent of its argument x.
- math.cos() Computes the cosine of its argument x, measured in radians.
- math.cosh() Computes the hyperbolic cosine of its argument x.
- math.sin() Computes the sine of its argument x, measured in radians.
- math.sinh() Computes the hyperbolic sine of its argument x.
- math.sqrt() Computes the square root of its argument x.
- math.tan() Computes the tangent of its argument x, measured in radians.
- math.tanh() Computes the hyperbolic tangent of its argument x.

Miscellaneous

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

- addr.extract_bits() Extracts bit_count bits (at most 32) starting with the bit number start_bit from the given IPv4 or IPv6 form of a non-negative integer.
- addr.is ipv4() Returns true if the address family of the given address is IPv4.
- addr.is_ipv6() Returns true if the address family of the given address is IPv6.
- http://newsletches.line.org/ Determines whether the HTTP status matches or does not match any of the statuses in the supplied in the supplied in the status matches.
- if() Implements a ternary operator for strings; if the expression is true, it returns value-when-true; if the expression is false,

- <u>setcookie.get_value_by_name()</u> Returns a value associated with the <u>cookie_name</u> in the <u>set-cookie</u> header contained in t where.
- std.collect() Combines multiple instances of the same header into one.
- <u>subfield()</u> Provides a means to access subfields from a header like <u>Cache-Control</u>, <u>Cookie</u>, and <u>Edge-Control</u> or individu string.

Query string manipulation

Fastly provides a number of extensions to VCL, including several functions for query-string manipulation based on Dridi Boukelmour

- <u>boltsort.sort()</u> Alias of querystring.sort.
- guerystring.add() Returns the given URL with the given parameter name and value appended to the end of the guery string.
- <u>querystring.clean()</u> Returns the given URL without empty parameters.
- <u>guerystring.filter_except()</u> Returns the given URL but only keeps the listed parameters.
- <u>querystring.filter()</u> Returns the given URL without the listed parameters.
- querystring.filtersep() Returns the separator needed by the querystring.filter() and querystring.filter_except() fu
- <u>querystring.globfilter_except()</u> Returns the given URL but only keeps the parameters matching a glob.
- <u>querystring.globfilter()</u> Returns the given URL without the parameters matching a glob.
- <u>querystring.regfilter_except()</u> Returns the given URL but only keeps the parameters matching a regular expression.
- <u>querystring.regfilter()</u> Returns the given URL without the parameters matching a regular expression.
- <u>querystring.remove()</u> Returns the given URL with its query-string removed.
- querystring.set() Returns the given URL with the given parameter name set to the given value, replacing the original value an
- <u>querystring.sort()</u> Returns the given URL with its query-string sorted.

<u>Randomness</u>

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

- randombool seeded() Identical to randombool, except takes an additional parameter, which is used to seed the random nun
- <u>randombool()</u> Returns a random, boolean value.
- randomint seeded() Identical to randomint, except takes an additional parameter used to seed the random number generate
- <u>randomint()</u> Returns a random integer value between from and to, inclusive.
- randomstr() Returns a random string of length [len] containing characters from the supplied string characters].

String manipulation

These functions provide various manipulation for strings containing arbitrary text content.

- <u>cstr_escape()</u> Escapes bytes from a string using C-style escape sequences.
- <u>json.escape()</u> Escapes characters of a UTF-8 encoded Unicode string using JSON-style escape sequences.
- regsub() Replaces the first occurrence of pattern, which is a Perl-compatible regular expression, in [input] with [replacements]
- regsuball() Replaces all occurrences of pattern, which may be a Perl-compatible regular expression, in [input] with repla
- std.anystr2ip() Converts the string addr to an IP address (IPv4 or IPv6).
- std.atof() Takes a string (which represents a float) as an argument and returns its value.
- <u>std.atoi()</u> Takes a string (which represents an integer) as an argument and returns its value.
- std.ip() An alias of std.str2ip().
- std.ip2str() Converts the IP address (v4 or v6) to a string.

- <u>std.prefixof()</u> True if the string s begins with the string begins with.
- std.str2ip() Converts the string representation of an IP address (IPv4 or IPv6) into an IP type.
- std.strlen() Returns the length of the string.
- std.strpad() This function constructs a string containing the input string s padded out with pad to produce a string of the g
- std.strrep() Repeats the given string n times.
- <u>std.strrev()</u> Reverses the given string.
- std.strstr() Returns the part of haystack string starting from and including the first occurrence of needle until the end of h
- <u>std.strtof()</u> Converts the string s to a float value with the given base base.
- std.strtol() Converts the string s to an integer value.
- std.suffixof() True if the string s ends with the string ends_with.
- std.tolower() Changes the case of a string to lowercase.
- std.toupper() Changes the case of a string to upper case.
- <u>substr()</u> Returns a substring of the byte string s, starting from the byte *offset*, of byte *length*.
- <u>urldecode()</u> Decodes a percent-encoded string.
- <u>urlencode()</u> Encodes a string for use in a URL.
- utf8.codepoint count() Returns the number of UTF-8 encoded Unicode code points in the string s.
- utf8.is_valid() Returns true if the string s contains valid UTF-8 and returns false if it does not contain valid UTF-8.
- utf8.strpad() Like std.strpad() except count gives the number of unicode code points for the output string rather than by
- utf8.substr() Returns a substring of the UTF-8 string s, starting from the Unicode code point offset, of Unicode code point le

Table

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

• table.lookup() — Look up the key key in the table ID.

TLS and HTTP/2

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

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

UUID

The universally unique identifier (UUID) module provides interfaces for generating and validating unique identifiers as defined by RFC on current time and host identity, are currently not supported.

- uuid.dns() Returns the RFC4122 identifier of DNS namespace, namely the constant ["6ba7b810-9dad-11d1-80b4-00c04fd43]
- <u>uuid.is_valid()</u> Returns true if the string holds a textual representation of a valid UUID (per RFC4122).
- <u>uuid.is_version3()</u> Returns true if string holds a textual representation of a valid version 3 UUID.
- <u>uuid.is_version4()</u> Returns true if string holds a textual representation of a valid version 4 UUID.
- <u>uuid.is_version5()</u> Returns true if string holds a textual representation of a valid version 5 UUID.
- <u>uuid.oid()</u> Returns the RFC4122 identifier of ISO OID namespace, namely the constant ["6ba7b812-9dad-11d1-80b4-00c04f
- <u>uuid.url()</u> Returns the RFC4122 identifier of URL namespace, namely the constant ["6ba7b811-9dad-11d1-80b4-00c04fd430
- <u>uuid.version3()</u> Derives a UUID corresponding to name within the given namespace using MD5 hash function.
- <u>uuid.version4()</u> Returns a UUID based on random number generator output.

- <u>uuid.version5()</u> Derives a UUID corresponding to name within the given namespace using SHA-1 hash function.
- uuid.x500() Returns the RFC4122 identifier of X.500 namespace, namely the constant ["6ba7b812-9dad-11d1-80b4-00c04fc

Variables

These VCL variables are supported by Fastly.

Date and time

By default VCL includes the now variable, which provides the current time (for example, Mon, 02 Jan 2006 22:04:05 GMT). Fastly and functions that allow more flexibility when dealing with dates and times.

- now.sec Like the now variable, but in seconds since the UNIX Epoch.
- now The current time in RFC 1123 format (e.g., Mon, 02 Jan 2006 22:04:05 GMT).
- time.elapsed.msec frac The time that has elapsed in milliseconds since the request started.
- time.elapsed.msec The time since the request start in milliseconds.
- <u>time.elapsed.sec</u> The time since the request start in seconds.
- time.elapsed.usec_frac The time the request started in microseconds since the last whole second.
- <u>time.elapsed.usec</u> The time since the request start in microseconds.
- <u>time.elapsed</u> The time since the request started.
- time.end.msec frac The time the request started in milliseconds since the last whole second.
- <u>time.end.msec</u> The time the request ended in milliseconds since the UNIX Epoch.
- <u>time.end.sec</u> The time the request ended in seconds since the UNIX Epoch.
- time.end.usec_frac The time the request started in microseconds since the last whole second.
- time.end.usec The time the request ended in microseconds since the UNIX Epoch.
- time.end The time the request ended, using RFC 1123 format (e.g., Mon, 02 Jan 2006 22:04:05 GMT).
- time.start.msec frac The time the request started in milliseconds since the last whole second, after TLS termination.
- time.start.msec The time the request started in milliseconds since the UNIX Epoch, after TLS termination.
- time.start.sec The time the request started in seconds since the UNIX Epoch, after TLS termination.
- time.start.usec frac The time the request started in microseconds since the last whole second, after TLS termination.
- time.start.usec The time the request started in microseconds since the UNIX Epoch, after TLS termination.
- time.start The time the request started, after TLS termination, using RFC 1123 format (e.g., Mon, 02 Jan 2006 22:04:05 G
- time.to first byte The time interval since the request started up to the point before the vcl deliver function ran.

Edge Side Includes (ESI)

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

- <u>req.esi</u> Whether or not to disable or enable ESI processing during this request.
- req.topurl In an ESI subrequest, contains the URL of the top-level request.

<u>Geolocation</u>

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

- <u>client.as.name</u> The name of the organization associated with client.as.number.
- <u>client.as.number</u> Autonomous system (AS) number.

- <u>client.geo.area_code</u> The telephone area code associated with the IP address.
- <u>client.geo.city.ascii</u> City or town name, encoded using ASCII encoding.
- <u>client.geo.city.latin1</u> City or town name, encoded using Latin-1 encoding.
- <u>client.geo.city.utf8</u> City or town name, encoded using UTF-8 encoding.
- <u>client.geo.city</u> Alias of client.geo.city.ascii.
- <u>client.geo.conn_speed</u> Connection speed.
- <u>client.geo.continent_code</u> Two-letter code representing the continent.
- <u>client.geo.country_code</u> A two-character ISO 3166-1 country code for the country associated with the IP address.
- <u>client.geo.country_code3</u> A three-character ISO 3166-1 alpha-3 country code for the country associated with the IP address
- <u>client.geo.country_name.ascii</u> Country name, encoded using ASCII encoding.
- <u>client.geo.country_name.latin1</u> Country name, encoded using Latin-1 encoding.
- <u>client.geo.country_name.utf8</u> Country name, encoded using UTF-8 encoding.
- <u>client.geo.country name</u> Alias of client.geo.country name.ascii.
- <u>client.geo.gmt_offset</u> Time zone offset from coordinated universal time (UTC) for client.geo.city.
- <u>client.geo.ip</u> <u>override</u> Override the IP address for geolocation data.
- <u>client.geo.latitude</u> Latitude, in units of degrees from the equator.
- client.geo.longitude Longitude, in units of degrees from the IERS Reference Meridian.
- client.geo.metro code Metro code.
- <u>client.geo.postal_code</u> The postal code associated with the IP address.
- <u>client.geo.region.ascii</u> ISO 3166-2 country subdivision code.
- <u>client.geo.region.latin1</u> Region code, encoded using Latin-1 encoding.
- <u>client.geo.region.utf8</u> Region code, encoded using UTF-8 encoding.
- <u>client.geo.region</u> Alias of client.geo.region.ascii.

Math constants and limits

Features that support various math constants and limits.

- math.1 PI The value of the reciprocal of math.PI (1/Pi).
- math.2 Pl The value of two times the reciprocal of math.Pl (2/Pi).
- math.2 SQRTPI The value of two times the reciprocal of the square root of math.PI (2/sqrt(Pi)).
- math.2PI The value of math.PI multiplied by two (Tau).
- math.E The value of the base of natural logarithms (e).
- math.FLOAT DIG Number of decimal digits that can be stored without loss in the FLOAT type.
- math.FLOAT EPSILON Minimum positive difference from 1.0 for the FLOAT type.
- math.FLOAT MANT DIG Number of hexadecimal digits stored for the significand in the [FLOAT] type.
- math.FLOAT MAX 10 EXP Maximum value in base 10 of the exponent part of the FLOAT type.
- math.FLOAT MAX EXP Maximum value in base 2 of the exponent part of the FLOAT type.
- math.FLOAT MAX Maximum finite value for the FLOAT type.
- math.FLOAT MIN 10 EXP Minimum value in base 10 of the exponent part of the FLOAT type.
- math.FLOAT MIN EXP Minimum value in base 2 of the exponent part of the [FLOAT] type.

- math.FLOAT MIN Minimum finite value for the FLOAT type.
- math.INTEGER BIT Number of bits in the INTEGER type.
- math.INTEGER MAX Maximum value for the INTEGER type.
- math.INTEGER MIN Minimum value for the INTEGER type.
- math.LN10 The value of the natural logarithm of 10 (log_e 10).
- math.LN2 The value of the natural logarithm of 2 (log_e 2).
- math.LOG10E The value of the logarithm to base 10 of math.E (log_10 e).
- math.LOG2E The value of the logarithm to base 2 of math.E (log_2 e).
- math.NAN A value that is "not a number." When converted to a STRING value, this is rendered as Nan.
- math.NEG HUGE VAL Negative overflow value.
- math.NEG INFINITY A value representing negative infinity (-∞).
- math.PHI The golden ratio (Φ).
- math.Pl 2 The value of math.Pl divided by two (Pi/2).
- math.Pl 4 The value of math.Pl divided by four (Pi/4).
- math.Pl The value of the ratio of a circle's circumference to its diameter (Pi).
- math.POS HUGE VAL Positive overflow value.
- math.POS INFINITY A value representing positive infinity (+∞).
- math.SQRT1 2 The value of the reciprocal of the square root of two (1/sqrt(2)).
- math.SQRT2 The value of the square root of two (sqrt(2)).
- math.TAU The value of math.PI multiplied by two (Tau).

Miscellaneous

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

- bereq.url.basename Same as req.url.basename, except for use between Fastly and your origin servers.
- bereq.url.dirname Same as req.url.dirname, except for use between Fastly and your origin servers.
- <u>bereq.url.qs</u> The query string portion of bereq.url.
- bereg.url The URL sent to the backend.
- beresp.backend.ip The IP of the backend this response was fetched from (backported from Varnish 3).
- beresp.backend.name The name of the backend this response was fetched from (backported from Varnish 3).
- <u>beresp.backend.port</u> The port of the backend this response was fetched from (backported from Varnish 3).
- <u>beresp.grace</u> Defines how long an object can remain overdue and still have Varnish consider it for grace mode.
- beresp.hipaa Specifies that content not be cached in non-volatile memory to help customers meet HIPAA security requirements
- <u>beresp.pci</u> Specifies that content be cached in a manner that satisfies PCI DSS requirements.
- <u>client.ip</u> The IP address of the client making the request.
- <u>client.port</u> Returns the remote client port.
- client.requests Tracks the number of requests received by Varnish over a persistent connection.
- <u>client.socket.pace</u> Ceiling rate in kilobytes per second for bytes sent to the client.
- <u>fastly.error</u> Contains the error code raised by the last function, otherwise *not set*.
- req.backend.healthy Whether or not this backend, or recursively any of the backends under this director, is considered healt

- reg.backend.is cluster True if this backend, or recursively any of the backends under this director, is a cluster backend.
- reg.backend.is origin True if this backend, or recursively any of the backends under this director, is not a shield backend.
- reg.backend.is shield True if this backend, or recursively any of the backends under this director, is a shield backend.
- <u>req.backend</u> The backend to use to service the request.
- req.body.base64 Same as req.body, except the request body is encoded in Base64, which handles null characters and allohodies.
- req.body The request body.
- reg.grace Defines how long an object can remain overdue and still have Varnish consider it for grace mode.
- req.http.host The full host name, without the path or query parameters.
- req.is ipv6 Indicates whether the request was made using IPv6 or not.
- reg.restarts Counts the number of times the VCL has been restarted.
- req.url.basename The file name specified in a URL.
- reg.url.dirname The directories specified in a URL.
- req.url.ext The file extension specified in a URL.
- req.url.path The full path, without any guery parameters.
- req.url.qs The query string portion of req.url.
- <u>req.url</u> The full path, including query parameters.
- stale.exists Specifies if a given object has stale content in cache.

Segmented Caching

Variables related to controlling range requests via Segmented Caching.

- <u>fastly.segmented_caching.autopurged</u> Whether an inconsistency encountered during Segmented Caching processing led to enqueuing a purge request.
- fastly.segmented caching.block number A zero-based counter identifying the file fragment being processed.
- <u>fastly.segmented_caching.cancelled</u> Whether Segmented Caching processing was enabled and cancelled by a non-206 res
- fastly.segmented caching.client req.is open ended Whether the client's request leaves the upper bound of the range open
- <u>fastly.segmented caching.client req.is range</u> Whether the client's request is a range request.
- fastly.segmented caching.client req.range high The upper bound of the client's requested range.
- <u>fastly.segmented_caching.client_req.range_low</u> The lower bound of the client's requested range.
- <u>fastly.segmented caching.completed</u> Whether Segmented Caching processing was enabled and cancelled by a non-206 re-
- <u>fastly.segmented_caching.error</u> The reason why Segmented Caching processing failed.
- fastly.segmented caching.failed Whether Segmented Caching processing was enabled and ended in a failure.
- <u>fastly.segmented caching is inner req</u> Whether VCL is running in the context of a sub-request that is retrieving a fragment of
- <u>fastly.segmented_caching.is_outer_reg</u> Whether VCL is running in the context of a request that is assembling file fragments
- <u>fastly.segmented_caching.obj.complete_length</u> The size of the whole file in bytes.
- fastly.segmented caching.rounded reg.range high The upper bound of the rounded block being processed.
- fastly.segmented caching.rounded req.range low The lower bound of the rounded block being processed.
- fastly.segmented_caching.total_blocks The number of fragments needed for assembling this response.

Server

Variables relating to the server receiving the request.

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

<u>Size</u>

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

- <u>bereq.body bytes written</u> Total body bytes written to a backend.
- <u>bereq.header bytes written</u> Total header bytes written to a backend.
- reg.body bytes read Total body bytes read from the client generating the request.
- reg.bytes read Total bytes read from the client generating the request.
- reg.header bytes read Total header bytes read from the client generating the request.
- resp.body bytes written Body bytes to send to the client in the response.
- <u>resp.bytes written</u> Total bytes to send to the client in the response.
- resp.completed Whether the response completed successfully or not.
- resp.header bytes written How many bytes were written for the header of a response.

TLS and HTTP/2

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

- fastly_info.h2.is_push Whether or not this request was a server-initiated request generated to create an HTTP/2 Server-push
- fastly_info.h2.stream_id If the request was made over HTTP/2, the underlying HTTP/2 stream ID.
- fastly_info.is_h2 Whether or not the request was made using HTTP/2.
- tls.client.cipher The cipher suite used to secure the client TLS connection.
- tls.client.ciphers list sha A SHA-1 digest of the raw buffer containing the list of supported ciphers, represented in Base64.
- <u>tls.client.ciphers list txt</u> The list of ciphers supported by the client, rendered as text, in a colon-separated list.
- <u>tls.client.ciphers</u> <u>list</u> The list of ciphers supported by the client, as sent over the network, hex encoded.
- tls.client.ciphers sha A SHA-1 of the cipher suite identifiers sent from the client as part of the TLS handshake, represented i
- <u>tls.client.protocol</u> The TLS protocol version this connection is speaking over.
- tls.client.servername The Server Name Indication (SNI) the client sent in the ClientHello TLS record.
- tls.client.tlsexts list sha A SHA-1 digest of the TLS extensions supported by the client as little-endian, 16-bit integers, repre
- tls.client.tlsexts list txt The list of TLS extensions supported by the client, rendered as text in a colon-separated list.
- tls.client.tlsexts_list The list of TLS extensions supported by the client as little-endian, 16-bit, unsigned integers, hex encode
- tls.client.tlsexts sha A SHA-1 of the TLS extension identifiers sent from the client as part of the TLS handshake, represented

Local variables

Fastly VCL supports variables for storing temporary values during request processing.

★ TIP: Consider using a req.http.* header to store a value if you need to pass information between functions or to the origin.

Declaring a variable

Variables must be declared before they are used, usually at the beginning of a function before any statements. They can only be use are declared. Fastly VCL does not provide block scope. Declarations apply to an entire function's scope even if a variable is declared.

Variables start with var. and their names consist of characters in the set [A-Za-z0-9._-1]. (: is explicitly disallowed.) The declara

```
declare local var.<name> <type>;
```

Variable types

Variables can be of the following types:

- BOOL
- FLOAT
- <u>INTEGER</u>
- <u>IP</u>
- RTIME (relative time)
- STRING
- TIME (absolute time)

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

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

Usage

Boolean variables

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

```
declare local var.boolean BOOL;
 2
 3 # BOOL assignment with RHS variable
 4 set var.boolean = true;
 5 set req.esi = var.boolean;
 6 set resp.http.Bool = if(req.esi, "y", "n");
 7
 8 # BOOL assignment with RHS function
   set var.boolean = http_status_matches(resp.status, "200,304");
 9
10
11 # BOOL assigment with RHS conditional
12 set var.boolean = (req.url == "/");
13
14 # 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 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.

```
1
   declare local var.integer INTEGER;
   declare local var.float FLOAT;
 2
 3
   # Numeric assignment with RHS variable and
 4
 5 # implicit string conversion for header
 6 set var.integer = req.bytes_read;
 7
   set var.integer -= req.body_bytes_read;
 8 set resp.http.VarInteger = var.integer;
10
   # Numeric comparison with RHS variable
11
   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.

```
1 declare local var.restarted STRING;
2
3 # String concatenation on RHS
4 set var.restarted = "Request " if(req.restarts > 0, "has", "has not") " restarted.";
```

IP address variables

IP address variables represent individual IP addresses.

```
1
    acl office_ip_ranges {
 2
      "192.0.2.0"/24;
                                                 # internal office
     "198.51.100.4";
 3
                                                 # remote VPN office
 4
      "2001:db8:ffff:ffff:ffff:ffff:ffff; # ipv6 address remote
 5
   }
 6
 7
    declare local var.ip1 IP;
 8
    set var.ip1 = "192.0.2.0";
 9
    if (var.ip1 ~ office_ip_ranges) {
10
11
    }
12
13
14
    declare local var.ip2 IP;
   set var.ip2 = "2001:db8:ffff:ffff:ffff:ffff:ffff;;;
```

Time variables

Time variables support both relative and absolute times.

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

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

set var.time = std.time("Fri, 10 Jun 2016 00:02:12 GMT", now);
set var.time -= var.rtime;

# implicit string conversion for header
set resp.http.VarTime = var.time;
```

Operators

Fastly VCL provides various arithmetic and conditional operators. Operators are syntactic items which evaluate to a value. Syntax is 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. I grammatical clauses are used in this document to indicate which types are valid operands. These are not precisely defined until the especified, 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 be Operators are listed in the following table as the highest precedence first. For example, a | b && c reads as a | (b && c) bec than | | |.

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

Operator	Name	Associativity
	Grouping for precedence	left to right
1	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

```
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 staten 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 its

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 pc either operand is infinity. However some situations evaluate to NaN instead. Some of these situations are *domain errors*, in which cas accordingly. Others situations are not domain errors: $\infty - \infty$ and $0 \times \infty$. These evaluate to NaN but do not set <u>fastly.error</u>.

Assignment

Assignment is provided by the = operator:

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

Addition and subtraction

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

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

Multiplication, division and modulus

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

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

Bitwise operators

```
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 ";"</pre>
```

Right shifts sign-extend negative numbers. For example, $-32 \gg 5$ gives -1.

Shift and rotate operations with negative shift widths perform the operation in the opposite direction. For example, 32 << -5 gives 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 $\left[\alpha \, \alpha\right]$ and $\left[\cdot \right]$ 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:

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

Boolean AND and OR operators (& a 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 example, given a && b when the left-hand operand is false, the resulting value will always be false, regardless of the value of the rig situation, the right-hand operand will not be evaluated. This can be seen when the right-hand operand has a visible side effect, such performs some action.

Comparison operators

FLOAT comparisons have special cases for operands which are NaN: The [] operator always evaluates to true when either operand operators always evaluate to false when either operand is NaN. For example, if a given variable is NaN, that variable will compare un 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 <code>!=</code> and <code>!~</code> operators always evaluate to false when either operand is not set. For example, if a given var compare unequal to itself: both <code>req.http.unset == req.http.unset</code> and <code>req.http.unset ~ ".?"</code> will be false.

Floating point infinities are signed, and compare as beyond the maximum and minimum values for FLOAT types, such that for any fir

The comparison operators are:

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

Equality is defined for all types:

```
:= 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 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: * & | >> << ++ -- %



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 conversion is not described except for types where it differs from the corresponding literal syntax.

- BOOL
- FLOAT
- INTEGER
- IP
- RTIME
- STRING
- TIME

Types with special semantics

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

- BACKEND
- HASH
- HEADER
- VOID

Directors

Fastly's directors contain a list of backends to direct requests to. Traffic is distributed according to the specific director policy.

Healthcheck probes should be defined for backends within directors so the director can check the backend health state before send 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 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 diagram 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 mus weight (~ 66%) to exceed the 50% quorum weight and qualify the director as healthy. If only one backend is healthy and the 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 rand three times before indicating all backends are unhealthy.

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 recrequest to $[F_backend1]$, and so on.

```
1 director my_dir round-robin {
2      { .backend = F_backend1; }
3      { .backend = F_backend2; }
4      { .backend = F_backend3; }
5 }
```

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 director will send all requests to F_backend2 until F_backend1 is healthy again. If F_backend1 and F_backend2 both become unhoused all requests to F_backend3 until either one of the previous backends become healthy again.

```
1 director my_dir fallback {
2      { .backend = F_backend1; }
3      { .backend = F_backend2; }
4      { .backend = F_backend3; }
5 }
```

Rounding modes

Fastly VCL provides access to various *rounding modes* by way of independent functions for rounding values. These functions have ϵ stateful interface to set a "current" rounding mode.

Fastly VCL does not provide interfaces to round values to a given number of significant figures, to a given multiple, or to a given pow

Tie-breaking when rounding to nearest

The roundoff errors introduced by rounding values to their nearest integers are symmetric, except for treatment of the exact midpoin

That is, for every value that gets rounded up (such as 3.77 rounding up to the nearest integer 4.0), there is a corresponding value (3.2 same amount. This can be seen visually:

Rounding to the nearest integer requires a tie-breaking rule for when the fractional part of a value is exactly 0.5. There are several wa in the "to nearest" rounding modes below.

Overview

Example values:

Input	ceil	floor	trunc	round	roundeven	roundhalfup	roundhalfdown
-1.8	-1.0	-2.0	-1.0	-2.0	-2.0	-2.0	-2.0
-1.5	-1.0	-2.0	-1.0	-2.0	-2.0	-1.0	-2.0
-1.2	-1.0	-2.0	-1.0	-1.0	-1.0	-1.0	-1.0
-0.5	-0.0	-1.0	-0.0	-1.0	-0.0	-0.0	-1.0
0.5	1.0	0.0	0.0	1.0	0.0	1.0	0.0
1.2	2.0	1.0	1.0	1.0	1.0	1.0	1.0
1.5	2.0	1.0	1.0	2.0	2.0	2.0	1.0
1.8	2.0	1.0	1.0	2.0	2.0	2.0	2.0

A visual representation of the same:

⟨— — -1.8 -1.5 -1.2 -0.5 0.5 1.2 1.5 1.8 "Direct" modes: math.ceil —										
"Direct" modes: math.ceil		‹ —		>			‹ —		 >	
math.ceil -> -> -> -> -> -> -> -> math.floor math.trunc "To nearest" modes: math.round math.round math.roundeven		-1.8	-1.5	-1.2	-0 . 5	0.5	1.2	1.5	1.8	
math.floor <	"Direct" modes:									
math.trunc -> -> -> - <	math.ceil	>	>	 >	 >	 >	 >	 >	 →	
"To nearest" modes: math.round	math.floor	<	<	<	<	<	<	<	‹ —	
$\begin{array}{cccccccccccccccccccccccccccccccccccc$	math.trunc	>	>	>	 →	<	<	<		
	"To nearest" modes:									
	math.round	<	<	>	<	>	<	>	 >	
mark many displayers	math.roundeven	<	<	>	 >	<	<	>	>	
·	math.roundhalfup	<	>	>	>	 >	<	 >	→ >	
	math.roundhalfdown	<	<	>	<	‹ —	<	<	<	

"Direct" rounding modes

• Round up — math.ceil()

Also known as ceiling, round towards positive infinity

IEEE 754 roundTowardPositive

Non-integer values are rounded up towards $+\infty$. Negative results thus round toward zero.

• Round down — math.floor()

Also known as floor, round towards negative infinity

IEEE 754 roundTowardNegative

Non-integer values are rounded down towards -∞. Negative results thus round away from zero.

• Round towards zero — math.trunc()

Also known as truncation, round away from infinity

IEEE 754 roundTowardZero

Rounding is performed by removing the fractional part of a number, leaving the integral part unchanged.

NOTE: The FLOAT to INTEGER type conversion in Fastly VCL is not by truncation (as it is in many comparable languages away from zero.

· Round away from zero

Also known as round towards infinity

Positive non-integer values are rounded up towards positive infinity. Negative non-integer values are rounded down towards no Not provided in Fastly VCL.

"To nearest" rounding modes

All of the following modes round non-tie values to their nearest integer. These modes differ only in their treatment of ties.

Round to nearest, ties away from zero — math.round()

Also known as commercial rounding

IEEE 754 roundTiesToAway

For positive values, ties are rounded up towards positive infinity. For negative values, ties are rounded down towards negative infinity. This is symmetric behavior, avoiding bias to either positive or negative values. However, this mode does introduce bias away from the control of the cont

This rounding mode is used for implicit FLOAT to INTEGER type conversions in VCL. These behave as if by a call to math.rou

Round to nearest, ties to even — <u>math.roundeven()</u>

Also known as half to even, convergent rounding, statistician's rounding, Dutch rounding, Gaussian rounding, odd-even rounding, IEEE 754 roundTiesToEven

Of the two nearest integer values, ties are rounded either up or down to whichever value is even.

This rounding mode increases the probability of even numbers relative to odd numbers, but avoids bias to either positive or netwards or away from zero. The cumulative error is minimized when summing rounded values, especially when the values are predominantly negative.

Round to nearest, ties towards positive infinity — <u>math.roundhalfup()</u>

Also known as half up

This is asymmetric behavior, where ties for negative values are rounded towards zero, and ties for positive values are rounded towards zero, and ties for positive values are rounded towards.

▲ WARNING: Some languages use the term *half up* to mean symmetric behavior. For rounding functions in these language absolute magnitude. That is, negative ties will be rounded away from zero, which differs from the behavior in VCL. Take care rounding mode to VCL.

Round to nearest, ties towards negative infinity — math.roundhalfdown()

Also known as half down

This is asymmetric behavior, where ties for negative values are rounded away from zero, and ties for positive values are rounded

▲ WARNING: Some languages use the term *half down* to mean symmetric behavior. For rounding functions in these languages absolute magnitude. That is, negative ties will be rounded towards zero, which differs from the behavior in VCL. Take care w rounding mode to VCL.

Round to nearest with other tie-breaking schemes

There are several other less common arrangements for tie-breaking. These include *ties to odd* (in a similar manner as *ties to ev* stochastic tie-breaking.

These schemes are not provided in Fastly VCL.

Floating point numbers have more computational nuances than are described by the cursory discussion of biases here. For more described by the cursory discussion of biases here. For more described by the cursory discussion of biases here. For more described by the cursory discussion of biases here. For more described by the cursory discussion of biases here.



Need some help?

Support portal

File a ticket

<u>Fastly status</u> <u>www.fastly.com</u> <u>Sitemap | Translations | Archives</u> Copyight © 2019 Fastly Inc. All Rights Reserved.

Policy FAQ | Acceptable Use | Terms of Service | Privacy.