HTTP M. Nottingham
Internet-Draft Fastly
Updates: 7234 (if approved) November 2, 2019
Intended status: Standards Track
Expires: May 5, 2020

HTTP Representation Variants
draft-ietf-httpbis-variants-06

Abstract

This specification introduces an alternative way to select a HTTP response from a cache based upon its request headers, using the HTTP "Variants" and "Variant-Key" response header fields. Its aim is to make HTTP proactive content negotiation more cache-friendly.

Note to Readers

RFC EDITOR: please remove this section before publication

Discussion of this draft takes place on the HTTP working group mailing list (ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/ [1].

Working Group information can be found at https://httpwg.github.io/ [2]; source code and issues list for this draft can be found at https://github.com/httpwg/http-extensions/labels/variants [3].

There is a prototype implementation of the algorithms herein at https://github.com/mnot/variants-toy [4].

Copyright Notice

Table of Contents

1. Introduction

HTTP proactive content negotiation ([RFC7231], Section 3.4.1) is seeing renewed interest, both for existing request headers like Accept-Language and for newer ones (for example, see [I-D.ietf-httpbis-client-hints]).

Successfully reusing negotiated responses that have been stored in a HTTP cache requires establishment of a secondary cache key ([RFC7234], Section 4.1). Currently, the Vary header ([RFC7231], Section 7.1.4) does this by nominating a set of request headers. Their values collectively form the secondary cache key for a given response.

HTTP's caching model allows a certain amount of latitude in normalising those request header field values, so as to increase the chances of a cache hit while still respecting the semantics of that header. However, normalisation is not formally defined, leading to infrequent implementation in cache, and divergence of behaviours when it is.

Even when the headers' semantics are understood, a cache does not know enough about the possible alternative representations available on the origin server to make an appropriate decision.

For example, if a cache has stored the following request/response pair:

```
GET /foo HTTP/1.1
Host: www.example.com
Accept-Language: en;q=0.5, fr;q=1.0


HTTP/1.1 200 OK
Content-Type: text/html
Content-Language: en
Vary: Accept-Language
Transfer-Encoding: chunked

[English content]
```

Provided that the cache has full knowledge of the semantics of Accept-Language and Content-Language, it will know that an English representation is available and might be able to infer that a French representation is not available. But, it does not know (for example) whether a Japanese representation is available without making another request, incurring possibly unnecessary latency.

This specification introduces the HTTP Variants response header field (Section 2) to enumerate the available variant representations on the origin server, to provide clients and caches with enough information to properly satisfy requests - either by selecting a response from cache or by forwarding the request towards the origin - by following the algorithm defined in Section 4.

Its companion Variant-Key response header field (Section 3) indicates the applicable key(s) that the response is associated with, so that it can be reliably reused in the future. Effectively, it allows the specification of a request header field to define how it affects the secondary cache key.

When this specification is in use, the example above might become:

```
GET /foo HTTP/1.1
Host: www.example.com
Accept-Language: en;q=0.5, fr;q=1.0
```

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Language: en
Vary: Accept-Language
Variants: Accept-Language;de;en;jp
Variant-Key: en
Transfer-Encoding: chunked

[English content]
```

Proactive content negotiation mechanisms that wish to be used with Variants need to define how to do so explicitly; see Section 6. As a result, it is best suited for negotiation over request headers that are well-understood.

Variants also works best when content negotiation takes place over a constrained set of representations; since each variant needs to be listed in the header field, it is ill-suited for open-ended sets of representations.

Variants can be seen as a simpler version of the Alternates header field introduced by [RFC2295]; unlike that mechanism, Variants does not require specification of each combination of attributes, and does not assume that each combination has a unique URL.

1.1. Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] but relies on Structured Headers from [I-D.ietf-httpbis-header-structure] for parsing.

Additionally, it uses the "field-name" rule from [RFC7230], "type", "subtype", "content-coding" and "language-range" from [RFC7231], and "cookie-name" from [RFC6265].

2. The "Variants" HTTP Header Field

The Variants HTTP response header field indicates what representations are available for a given resource at the time that the response is produced, by enumerating the request header fields that it varies on, along with a representation of the values that are available for each.

Variants is a Structured Header Dictionary (Section 3.2 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

```
Variants        = sh-dict
```

Each member-name represents the field-name of a request header that is part of the secondary cache key; each member-value is an inner-list of strings or tokens that convey representations of potential values for that header field, hereafter referred to as "available-values".

If Structured Header parsing fails or a member's value does have the structure outlined above, the client MUST treat the representation as having no Variants header field.

Note that an available-value that is a token is interpreted as a string containing the same characters, and vice versa.

So, given this example header field:

```
Variants: Accept-Encoding=(gzip)
```

a recipient can infer that the only content-coding available for that resource is "gzip" (along with the "identity" non-encoding; see Appendix A.2).

Given:

```
Variants: accept-encoding=()
```

a recipient can infer that no content-codings (beyond identity) are supported. Note that as always, field-name is case-insensitive.

A more complex example:

```
Variants: Accept-Encoding=(gzip br), Accept-Language=(en fr)
```

Here, recipients can infer that two content-codings in addition to "identity" are available, as well as two content languages. Note that, as with all Structured Header dictionaries, they might occur in the same header field or separately, like this:

```
Variants: Accept-Encoding=(gzip brotli)
Variants: Accept-Language=(en fr)
```

The ordering of available-values is significant, as it might be used by the header's algorithm for selecting a response (in this example, the first language is the default; see Appendix A.3).

The ordering of the request header fields themselves indicates descending application of preferences; in the example above, a cache that has all of the possible permutations stored will honour the client's preferences for Accept-Encoding before honouring Accept-Language.

Origin servers SHOULD consistently send Variant header fields on all cacheable (as per [RFC7234], Section 3) responses for a resource, since its absence will trigger caches to fall back to Vary processing.

Likewise, servers MUST send the Variant-Key response header field when sending Variants, since its absence means that the stored response will not be reused when this specification is implemented.

RFC EDITOR: Please remove the next paragraph before publication.

Implementations of drafts of this specification MUST implement an HTTP header field named "Variants-##" instead of the "Variants" header field specified by the final RFC, with "##" replaced by the

draft number being implemented. For example, implementations of draft-ietf-httpbis-variants-05 would implement "Variants-05".

2.1. Relationship to Vary

This specification updates [RFC7234] to allow caches that implement it to ignore request header fields in the Vary header for the purposes of secondary cache key calculation ([RFC7234], Section 4.1) when their semantics are implemented as per this specification and their corresponding response header field is listed in Variants.

If any member of the Vary header does not have a corresponding variant that is understood by the implementation, it is still subject to the requirements there.

See Section 5.1.3 for an example.

In practice, implementation of Vary varies considerably. As a result, cache efficiency might drop considerably when Variants does not contain all of the headers referenced by Vary, because some implementations might choose to disable Variants processing when this is the case.

3. The "Variant-Key" HTTP Header Field

The Variant-Key HTTP response header field identifies one or more sets of available-values that identify the secondary cache key(s) that the response it occurs within are associated with.

Variant-Key is a Structured Header List (Section 3.1 of [I-D.ietf-httpbis-header-structure]) whose members are inner-lists of strings or tokens. Its ABNF is:

```
Variant-Key      = sh-list
```

Each member MUST be an inner-list, and MUST itself have the same number of members as there are members of the representation's Variants header field. If not, the client MUST treat the representation as having no Variant-Key header field.

Each member identifies a list of available-values corresponding to the header field-names in the Variants header field, thereby identifying a secondary cache key that can be used with this response. These available-values do not need to explicitly appear in the Variants header field; they can be interpreted by the algorithm specific to processing that field. For example, Accept-Encoding defines an implicit "identity" available-value (Appendix A.2).

Each inner-list member is treated as identifying an available-value for the corresponding variant-axis' field-name. Any list-member that is a token is interpreted as a string containing the same characters.

For example:

```
Variants: Accept-Encoding=(gzip br), Accept-Language=(en fr)
Variant-Key: (gzip fr)
```

This header pair indicates that the representation has a "gzip" content-coding and "fr" content-language.

If the response can be used to satisfy more than one request, they can be listed in additional members. For example:

```
Variants: Accept-Encoding=(gzip br), Accept-Language=(en fr)
Variant-Key: (gzip fr), ("identity" fr)
```

indicates that this response can be used for requests whose Accept-Encoding algorithm selects "gzip" or "identity", as long as the Accept-Language algorithm selects "fr" - perhaps because there is no gzip-compressed French representation.

When more than one Variant-Key value is in a response, the first one present MUST correspond to the request that caused that response to be generated. For example:

```
Variants: Accept-Encoding=(gzip br), Accept-Language=(en fr)
Variant-Key: (gzip fr), (identity fr), (br fr oops)
```

is treated as if the Variant-Key header were completely absent, which will tend to disable caching for the representation that contains it.

Note that in

```
Variant-Key: (gzip  fr)
Variant-Key: ("gzip " fr)
```

The whitespace after "gzip" in the first header field value is excluded by the parsing algorithm, but the whitespace in the second header field value is included by the string parsing algorithm. This will likely cause the second header field value to fail to match client requests.

RFC EDITOR: Please remove the next paragraph before publication.

Implementations of drafts of this specification MUST implement an HTTP header field named "Variant-Key-##" instead of the "Variant-Key"

header field specified by the final RFC, with "##" replaced by the draft number being implemented. For example, implementations of draft-ietf-httpbis-variants-05 would implement "Variant-Key-05".

4. Cache Behaviour

Caches that implement the Variants header field and the relevant semantics of the field-names it contains can use that knowledge to either select an appropriate stored representation, or forward the request if no appropriate representation is stored.

They do so by running this algorithm (or its functional equivalent) upon receiving a request:

Given incoming-request (a mapping of field-names to field-values, after being combined as allowed by Section 3.2.2 of [RFC7230]), and stored-responses (a list of stored responses suitable for reuse as defined in Section 4 of [RFC7234], excepting the requirement to calculate a secondary cache key):

1. If stored-responses is empty, return an empty list.

2. Order stored-responses by the "Date" header field, most recent to least recent.

3. Let sorted-variants be an empty list.

4. If the freshest member of stored-responses (as per [RFC7234], Section 4.2) has one or more "Variants" header field(s) that successfully parse according to Section 2:

    1. Select one member of stored-responses with a "Variants" header field-value(s) that successfully parses according to Section 2 and let variants-header be this parsed value. This SHOULD be the most recent response, but MAY be from an older one as long as it is still fresh.

    2. For each variant-axis in variants-header:

        1. If variant-axis' field-name corresponds to the request header field identified by a content negotiation mechanism that the implementation supports:

            1. Let request-value be the field-value associated with field-name in incoming-request, or null if field-name is not in incoming-request.

2. Let sorted-values be the result of running the algorithm defined by the content negotiation mechanism with request-value and variant-axis' available-values.

3. Append sorted-values to sorted-variants.

At this point, sorted-variants will be a list of lists, each member of the top-level list corresponding to a variant-axis in the Variants header field-value, containing zero or more items indicating available-values that are acceptable to the client, in order of preference, greatest to least.

5. Return result of running Compute Possible Keys (Section 4.1) on sorted-variants, an empty list and an empty list.

This returns a list of lists of strings suitable for comparing to the parsed Variant-Keys (Section 3) that represent possible responses on the server that can be used to satisfy the request, in preference order, provided that their secondary cache key (after removing the headers covered by Variants) matches. Section 4.2 illustrates one way to do this.

4.1. Compute Possible Keys

This algorithm computes the cross-product of the elements of key-facets.

Given key-facets (a list of lists of strings), and key-stub (a list of strings representing a partial key), and possible-keys (a list of lists of strings):

1. Let values be the first member of key-facets.

2. Let remaining-facets be a copy of all of the members of key-facets except the first.

3. For each value in values:

    1. Let this-key be a copy of key-stub.

    2. Append value to this-key.

    3. If remaining-facets is empty, append this-key to possible-keys.

    4. Otherwise, run Compute Possible Keys on remaining-facets, this-key and possible-keys.

4. Return possible-keys.

4.2. Check Vary

This algorithm is an example of how an implementation can meet the requirement to apply the members of the Vary header field that are not covered by Variants.

Given incoming-request (a mapping of field-names to field-values, after being combined as allowed by Section 3.2.2 of [RFC7230]), and stored-response (a stored response):

1. Let filtered-vary be the field-value(s) of stored-response's "Vary" header field.

2. Let processed-variants be a list containing the request header fields that identify the content negotiation mechanisms supported by the implementation.

3. Remove any member of filtered-vary that is a case-insensitive match for a member of processed-variants.

4. If the secondary cache key (as calculated in [RFC7234], Section 4.1) for stored_response matches incoming-request, using filtered-vary for the value of the "Vary" response header, return True.

5. Return False.

This returns a Boolean that indicates whether stored-response can be used to satisfy the request.

Note that implementation of the Vary header field varies in practice, and the algorithm above illustrates only one way to apply it. It is equally viable to forward the request if there is a request header listed in Vary but not Variants.

4.3. Example of Cache Behaviour

For example, if the selected variants-header was:

```
Variants: Accept-Language=(en fr de), Accept-Encoding=(gzip br)
```

and the request contained the headers:

```
Accept-Language: fr;q=1.0, en;q=0.1
Accept-Encoding: gzip
```

Then the sorted-variants would be:

```
[
  ["fr", "en"]         // prefers French, will accept English
  ["gzip", "identity"] // prefers gzip encoding, will accept identity
]
```

Which means that the result of the Cache Behaviour algorithm would be:

```
[
  ["fr", "gzip"],
  ["fr", "identity"],
  ["en", "gzip"],
  ["en", "identity"]
]
```

Representing a first preference of a French, gzip'd response. Thus, if a cache has a response with:

```
Variant-Key: (fr gzip)
```

it could be used to satisfy the first preference. If not, responses corresponding to the other keys could be returned, or the request could be forwarded towards the origin.

4.3.1. A Variant Missing From the Cache

If the selected variants-header was:

```
Variants: Accept-Language=(en fr de)
```

And a request comes in with the following headers:

```
Accept-Language: de;q=1.0, es;q=0.8
```

Then sorted-variants in Cache Behaviour is:

```
[
  ["de"]         // prefers German; will not accept English
]
```

If the cache contains responses with the following Variant-Keys:

```
Variant-Key: (fr)
Variant-Key: (en)
```

Then the cache needs to forward the request to the origin server, since Variants indicates that "de" is available, and that is acceptable to the client.

4.3.2. Variants That Don't Overlap the Client's Request

If the selected variants-header was:

```
Variants: Accept-Language=(en fr de)
```

And a request comes in with the following headers:

```
Accept-Language: es;q=1.0, ja;q=0.8
```

Then sorted-variants in Cache Behaviour are:

```
[
  ["en"]
]
```

This allows the cache to return a "Variant-Key: en" response even though it's not in the set the client prefers.

5. Origin Server Behaviour

Origin servers that wish to take advantage of Variants will need to generate both the Variants (Section 2) and Variant-Key (Section 3) header fields in all cacheable responses for a given resource. If either is omitted and the response is stored, it will have the effect of disabling caching for that resource until it is no longer stored (e.g., it expires, or is evicted).

Likewise, origin servers will need to assure that the members of both header field values are in the same order and have the same length, since discrepancies will cause caches to avoid using the responses they occur in.

The value of the Variants header should be relatively stable for a given resource over time; when it changes, it can have the effect of invalidating previously stored responses.

As per Section 2.1, the Vary header is required to be set appropriately when Variants is in use, so that caches that do not implement this specification still operate correctly.

Origin servers are advised to carefully consider which content negotiation mechanisms to enumerate in Variants; if a mechanism is

not supported by a receiving cache, it will "downgrade" to Vary handling, which can negatively impact cache efficiency.

5.1. Examples

The operation of Variants is illustrated by the examples below.

5.1.1. Single Variant

Given a request/response pair:

```
GET /clancy HTTP/1.1
Host: www.example.com
Accept-Language: en;q=1.0, fr;q=0.5


HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Cache-Control: max-age=3600
Variants: Accept-Language=(en de)
Variant-Key: (en)
Vary: Accept-Language
Transfer-Encoding: chunked
```

Upon receipt of this response, the cache knows that two representations of this resource are available, one with a language of "en", and another whose language is "de".

Subsequent requests (while this response is fresh) will cause the cache to either reuse this response or forward the request, depending on what the selection algorithm determines.

So, if a request with "en" in Accept-Language is received and its q-value indicates that it is acceptable, the stored response is used. A request that indicates that "de" is acceptable will be forwarded to the origin, thereby populating the cache. A cache receiving a request that indicates both languages are acceptable will use the q-value to make a determination of what response to return.

A cache receiving a request that does not list either language as acceptable (or does not contain an Accept-Language at all) will return the "en" representation (possibly fetching it from the origin), since it is listed first in the Variants list.

Note that Accept-Language is listed in Vary, to assure backwards-compatibility with caches that do not support Variants.

5.1.2. Multiple Variants

A more complicated request/response pair:

```
GET /murray HTTP/1.1
Host: www.example.net
Accept-Language: en;q=1.0, fr;q=0.5
Accept-Encoding: gzip, br


HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Content-Encoding: br
Variants: Accept-Language=(en jp de)
Variants: Accept-Encoding=(br gzip)
Variant-Key: (en br)
Vary: Accept-Language, Accept-Encoding
Transfer-Encoding: chunked
```

Here, the cache knows that there are two axes that the response varies upon; language and encoding. Thus, there are a total of nine possible representations for the resource (including the identity encoding), and the cache needs to consider the selection algorithms for both axes.

Upon a subsequent request, if both selection algorithms return a stored representation, it can be served from cache; otherwise, the request will need to be forwarded to origin.

5.1.3. Partial Coverage

Now, consider the previous example, but where only one of the Vary'd axes (encoding) is listed in Variants:

```
GET /bar HTTP/1.1
Host: www.example.net
Accept-Language: en;q=1.0, fr;q=0.5
Accept-Encoding: gzip, br
```

```
HTTP/1.1 200 OK
Content-Type: image/gif
Content-Language: en
Content-Encoding: br
Variants: Accept-Encoding=(br gzip)
Variant-Key: (br)
Vary: Accept-Language, Accept-Encoding
Transfer-Encoding: chunked
```

Here, the cache will need to calculate a secondary cache key as per [RFC7234], Section 4.1 - but considering only Accept-Language to be in its field-value - and then continue processing Variants for the set of stored responses that the algorithm described there selects.

6. Defining Content Negotiation Using Variants

To be usable with Variants, proactive content negotiation mechanisms need to be specified to take advantage of it. Specifically, they:

- MUST define a request header field that advertises the clients preferences or capabilities, whose field-name SHOULD begin with "Accept-".

- MUST define the syntax of an available-value that will occur in Variants and Variant-Key.

- MUST define an algorithm for selecting a result. It MUST return a list of available-values that are suitable for the request, in order of preference, given the value of the request header nominated above (or null if the request header is absent) and an available-values list from the Variants header. If the result is an empty list, it implies that the cache cannot satisfy the request.

Appendix A fulfils these requirements for some existing proactive content negotiation mechanisms in HTTP.

7. IANA Considerations

This specification registers the following entry in the Permanent Message Header Field Names registry established by [RFC3864]:

- Header field name: Variants

- Applicable protocol: http

- Status: standard

o Author/Change Controller: IETF

o Specification document(s): [this document]

o Related information:

This specification registers the following entry in the Permanent Message Header Field Names registry established by [RFC3864]:

o Header field name: Variant-Key

o Applicable protocol: http

o Status: standard

o Author/Change Controller: IETF

o Specification document(s): [this document]

o Related information:

8. Security Considerations

If the number or advertised characteristics of the representations available for a resource are considered sensitive, the Variants header by its nature will leak them.

Note that the Variants header is not a commitment to make representations of a certain nature available; the runtime behaviour of the server always overrides hints like Variants.

9. References

9.1. Normative References

[I-D.ietf-httpbis-header-structure]
Nottingham, M. and P. Kamp, "Structured Headers for HTTP", draft-ietf-httpbis-header-structure-13 (work in progress), August 2019.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC4647] Phillips, A. and M. Davis, "Matching of Language Tags", BCP 47, RFC 4647, DOI 10.17487/RFC4647, September 2006, <https://www.rfc-editor.org/info/rfc4647>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, DOI 10.17487/RFC6265, April 2011, <https://www.rfc-editor.org/info/rfc6265>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, DOI 10.17487/RFC7234, June 2014, <https://www.rfc-editor.org/info/rfc7234>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

9.2. Informative References

[I-D.ietf-httpbis-client-hints] Grigorik, I., "HTTP Client Hints", draft-ietf-httpbis-client-hints-07 (work in progress), March 2019.

[RFC2295] Holtman, K. and A. Mutz, "Transparent Content Negotiation in HTTP", RFC 2295, DOI 10.17487/RFC2295, March 1998, <https://www.rfc-editor.org/info/rfc2295>.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, DOI 10.17487/RFC3864, September 2004, <https://www.rfc-editor.org/info/rfc3864>.

9.3. URIs

[1] https://lists.w3.org/Archives/Public/ietf-http-wg/

[2] https://httpwg.github.io/

[3] https://github.com/httpwg/http-extensions/labels/variants

[4] https://github.com/mnot/variants-toy

Appendix A. Variants for Existing Content Negotiation Mechanisms

This appendix defines the required information to use existing proactive content negotiation mechanisms (as defined in [RFC7231], Section 5.3) with the Variants header field.

A.1. Accept

This section defines variant handling for the Accept request header (section 5.3.2 of [RFC7231]).

The syntax of an available-value for Accept is:

```
accept-available-value = type "/" subtype
```

To perform content negotiation for Accept given a request-value and available-values:

1. Let preferred-available be an empty list.

2. Let preferred-types be a list of the types in the request-value (or the empty list if request-value is null), ordered by their weight, highest to lowest, as per Section 5.3.2 of [RFC7231] (omitting any coding with a weight of 0). If a type lacks an explicit weight, an implementation MAY assign one.

3. For each preferred-type in preferred-types:

    1. If any member of available-values matches preferred-type, using the media-range matching mechanism specified in Section 5.3.2 of [RFC7231] (which is case-insensitive), append those members of available-values to preferred-available (preserving the precedence order implied by the media ranges' specificity).

4. If preferred-available is empty, append the first member of available-values to preferred-available. This makes the first available-value the default when none of the client's preferences are available.

5. Return preferred-available.

Note that this algorithm explicitly ignores extension parameters on media types (e.g., "charset").

A.2. Accept-Encoding

This section defines variant handling for the Accept-Encoding request header (section 5.3.4 of [RFC7231]).

The syntax of an available-value for Accept-Encoding is:

accept-encoding-available-value = content-coding / "identity"

To perform content negotiation for Accept-Encoding given a request-value and available-values:

1. Let preferred-available be an empty list.

2. Let preferred-codings be a list of the codings in the request-value (or the empty list if request-value is null), ordered by their weight, highest to lowest, as per Section 5.3.1 of [RFC7231] (omitting any coding with a weight of 0). If a coding lacks an explicit weight, an implementation MAY assign one.

3. If "identity" is not a member of preferred-codings, append "identity".

4. Append "identity" to available-values.

5. For each preferred-coding in preferred-codings:

    1. If there is a case-insensitive, character-for-character match for preferred-coding in available-values, append that member of available-values to preferred-available.

6. Return preferred-available.

Note that the unencoded variant needs to have a Variant-Key header field with a value of "identity" (as defined in Section 5.3.4 of [RFC7231]).

A.3. Accept-Language

This section defines variant handling for the Accept-Language request header (section 5.3.5 of [RFC7231]).

The syntax of an available-value for Accept-Language is:

accept-encoding-available-value = language-range

To perform content negotiation for Accept-Language given a request-value and available-values:

1. Let preferred-available be an empty list.

2. Let preferred-langs be a list of the language-ranges in the request-value (or the empty list if request-value is null), ordered by their weight, highest to lowest, as per Section 5.3.1 of [RFC7231] (omitting any language-range with a weight of 0). If a language-range lacks a weight, an implementation MAY assign one.

3. For each preferred-lang in preferred-langs:

    1. If any member of available-values matches preferred-lang, using either the Basic or Extended Filtering scheme defined in Section 3.3 of [RFC4647], append those members of available-values to preferred-available (preserving their order).

4. If preferred-available is empty, append the first member of available-values to preferred-available. This makes the first available-value the default when none of the client's preferences are available.

5. Return preferred-available.

A.4. Cookie

This section defines variant handling for the Cookie request header ([RFC6265]).

This syntax of an available-value for Cookie is:

```
cookie-available-value = cookie-name
```

To perform content negotiation for Cookie given a request-value and available-values:

1. Let cookies-available be an empty list.

2. For each available-value of available-values:

    1. Parse request-value as a Cookie header field [RFC6265] and let request-cookie-value be the cookie-value corresponding to a cookie with a cookie-name that matches available-value. If no match is found, continue to the next available-value.

    2. append request-cookie-value to cookies-available.

3. Return cookies-available.

A simple example is allowing a page designed for users that aren't logged in (denoted by the "logged_in" cookie-name) to be cached:

```
Variants: Cookie=(logged_in)
Variant-Key: (0)
Vary: Cookie
```

Here, a cache that implements Variants will only use this response to satisfy requests with "Cookie: logged_in=0". Caches that don't implement Variants will vary the response on all Cookie headers.

Or, consider this example:

```
Variants: Cookie=(user_priority)
Variant-Key: (silver), ("bronze")
Vary: Cookie
```

Here, the "user_priority" cookie-name allows requests from "gold" users to be separated from "silver" and "bronze" ones; this response is only served to the latter two.

It is possible to target a response to a single user; for example:

```
Variants: Cookie=(user_id)
Variant-Key: (some_person)
Vary: Cookie
```

Here, only the "some_person" "user_id" will have this response served to them again.

Note that if more than one cookie-name serves as a cache key, they'll need to be listed in separate Variants members, like this:

```
Variants: Cookie=(user_priority), Cookie=(user_region)
Variant-Key: (gold europe)
Vary: Cookie
```

Acknowledgements

This protocol is conceptually similar to, but simpler than, Transparent Content Negotiation [RFC2295]. Thanks to its authors for their inspiration.

It is also a generalisation of a Fastly VCL feature designed by Rogier 'DocWilco' Mulhuijzen.

Thanks to Hooman Beheshti, Ilya Grigorik, Leif Hedstrom, and Jeffrey Yasskin for their review and input.

Author's Address

Mark Nottingham
Fastly

Email: mnot@mnot.net
URI: https://www.mnot.net/