HyperText Transfer Protocol
Internet-Draft
Intended status: Standards Track
Expires: 2 April 2026

D. Denicola
J. Roman
Google LLC

29 September 2025

The No-Vary-Search HTTP Response Header Field
draft-ietf-httpbis-no-vary-search-03

Abstract

This specification defines a proposed HTTP response header field for changing how URL search parameters impact caching.

About This Document

This note is to be removed before publishing as an RFC.

The latest revision of this draft can be found at https://httpwg.org/http-extensions/draft-ietf-httpbis-no-vary-search.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-httpbis-no-vary-search/.

Discussion of this document takes place on the HTTP Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/. Working Group information can be found at https://httpwg.org/.

Source for this draft and an issue tracker can be found at https://github.com/httpwg/http-extensions/labels/no-vary-search.

Table of Contents

1. Introduction

HTTP caching [HTTP-CACHING] is based on reusing resources which match across a number of cache keys. One of the most prominent is the presented target URI (Section 7.1 of [HTTP]). However, sometimes multiple URLs can represent the same resource. This leads to caches not always being as helpful as they could be: if the cache contains the resource under one URI, but the resource is then requested under

another, the cached version will be ignored.

The No-Vary-Search HTTP header field tackles a specific subset of this general problem, for when a resource has multiple URLs which differ only in certain query components. It allows resources to declare that some or all parts of the query do not semantically affect the served resource, and thus can be ignored for cache matching purposes. For example, if the order of the query parameter keys do not semantically affect the served resource, this is indicated using

```
No-Vary-Search: key-order
```

If the specific query parameters (e.g., ones indicating something for analytics) do not semantically affect the served resource, this is indicated using

```
No-Vary-Search: params=("utm_source" "utm_medium" "utm_campaign")
```

And if the resource instead wants to take an allowlist-based approach, where only certain known query parameters semantically affect the served resource, they can use

```
No-Vary-Search: params, except=("productId")
```

Section 3 defines the header, using the [STRUCTURED-FIELDS] framework. Section 4 and Section 5 illustrate the data model for how the header can be represented in specifications, and the process for parsing the raw output from the structured field parser into that data model. Section 6 gives the key algorithm for comparing if two URLs are equivalent under the influence of the header; notably, it leans on the decomposition of the query component into keys and values given by the application/x-www-form-urlencoded (https://url.spec.whatwg.org/#concept-urlencoded) format specified in [WHATWG-URL]. (As such, this header is not useful for URLs whose query component does not follow that format.) Finally, Section 7 explains how to modify [HTTP-CACHING] to take into account this new equivalence.

2. Conventions and Definitions

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 document also adopts some conventions and notation typical in WHATWG and W3C usage, especially as it relates to algorithms. See [WHATWG-INFRA], and in particular:

* its definition of lists, including the list literal notation 「 1, 2, 3 」.

* its definition of strings, including their representation as code units.

(Other concepts used are called out using inline references.)

3. HTTP header field definition

The No-Vary-Search HTTP header field is a structured field [STRUCTURED-FIELDS] whose value MUST be a dictionary (Section 3.2 of [STRUCTURED-FIELDS]).

It has the following authoring conformance requirements:

* If present, the key-order entry's value MUST be a boolean (Section 3.3.6 of [STRUCTURED-FIELDS]).

* If present, the params entry's value MUST be either a boolean (Section 3.3.6 of [STRUCTURED-FIELDS]) or an inner list (Section 3.1.1 of [STRUCTURED-FIELDS]).

* If present, the except entry's value MUST be an inner list (Section 3.1.1 of [STRUCTURED-FIELDS]).

* The except entry MUST only be present if the params entry is also present, and the params entry's value is the boolean value true.

The dictionary MAY contain entries whose keys are not one of key-order, params, and except, but their meaning is not defined by this specification. Implementations of this specification will ignore such entries (but future documents might assign meaning to such entries).

> | As always, the authoring conformance requirements are not binding on implementations. Implementations instead need to implement the processing model given by the obtain a URL search variance algorithm (Section 5.2).

4. Data model

A _URL search variance_ consists of the following:

no-vary params
: either the special value *wildcard* or a list of strings

vary params
: either the special value *wildcard* or a list of strings

vary on key order
: a boolean

The _default URL search variance_ is a URL search variance whose no-vary params is an empty list, vary params is *wildcard*, and vary on key order is true.

The obtain a URL search variance algorithm (Section 5.2) ensures that all URL search variances obey the following constraints:

* vary params is a list if and only if the no-vary params is *wildcard*; and

* no-vary params is a list if and only if the vary params is *wildcard*.

5. Parsing

5.1. Parse a URL search variance

To _parse a URL search variance_ given _value_:

1. If _value_ is null, then return the default URL search variance.

2. Let _result_ be a new URL search variance.

3. Set _result_'s vary on key order to true.

4. If _value_["key-order"] exists:

    1. If _value_["key-order"] is not a boolean, then return the default URL search variance.

    2. Set _result_'s vary on key order to the boolean negation of _value_["key-order"].

5. If _value_["params"] exists:

    1. If _value_["params"] is a boolean:

        1. If _value_["params"] is true, then:

1. Set _result_’s no-vary params to *wildcard*.

2. Set _result_’s vary params to the empty list.

2. Otherwise:

1. Set _result_’s no-vary params to the empty list.

2. Set _result_’s vary params to *wildcard*.

2. Otherwise, if _value_["params"] is an array:

1. If any item in _value_["params"] is not a string, then return the default URL search variance.

2. Set _result_’s no-vary params to the result of applying parse a key (Section 5.3) to each item in _value_["params"].

3. Set _result_’s vary params to *wildcard*.

3. Otherwise, return the default URL search variance.

6. If _value_["except"] exists:

1. If _value_["params"] is not true, then return the default URL search variance.

2. If _value_["except"] is not an array, then return the default URL search variance.

3. If any item in _value_["except"] is not a string, then return the default URL search variance.

4. Set _result_’s vary params to the result of applying parse a key (Section 5.3) to each item in _value_["except"].

7. Return _result_.

| In general, this algorithm is strict and tends to return the default URL search variance whenever it sees something it doesn’t recognize. This is because the default URL search variance behavior will just cause fewer cache hits, which is an acceptable fallback behavior.
|

> | However, unrecognized keys at the top level are ignored, to make it easier to extend this specification in the future. To avoid misbehavior with existing client software, such extensions will likely expand, rather than reduce, the set of requests that a cached response can match.
>
> | The input to this algorithm is generally obtained by parsing a structured field (Section 4.2 of [STRUCTURED-FIELDS]) using field_type "dictionary".

5.2. Obtain a URL search variance

To _obtain a URL search variance_ given a response (https://fetch.spec.whatwg.org/#concept-response) _response_:

1. Let _fieldValue_ be the result of getting a structured field value (https://fetch.spec.whatwg.org/#concept-header-list-get-structured-header) [FETCH] given ‘No-Vary-Search‘ and "dictionary" from _response_’s header list.

2. Return the result of parsing a URL search variance (Section 5.1) given _fieldValue_.

5.2.1. Examples

The following illustrates how various inputs are parsed, in terms of their impacting on the resulting no-vary params and vary params:

Input	Result
No-Vary-Search: params	no-vary params: *wildcard* vary params: (empty list)
No-Vary-Search: params=("a")	no-vary params: 束 "a" 損 vary params: *wildcard*
No-Vary-Search: params, except=("x")	no-vary params: *wildcard* vary params: 束 "x" 損

Table 1

The following inputs are all invalid and will cause the default URL search variance to be returned:

* No-Vary-Search: unknown-key
* No-Vary-Search: key-order="not a boolean"

* No-Vary-Search: params="not a boolean or inner list"
* No-Vary-Search: params=(not-a-string)
* No-Vary-Search: params=("a"), except=("x")
* No-Vary-Search: params=(), except=()
* No-Vary-Search: params=?0, except=("x")
* No-Vary-Search: params, except=(not-a-string)
* No-Vary-Search: params, except="not an inner list"
* No-Vary-Search: params, except=?1
* No-Vary-Search: except=("x")
* No-Vary-Search: except=()

The following inputs are valid, but somewhat unconventional. They are shown alongside their more conventional form.

Input	Conventional form
No-Vary-Search: params=?1	No-Vary-Search: params
No-Vary-Search: key-order=?1	No-Vary-Search: key-order
No-Vary-Search: params, key-order, except=("x")	No-Vary-Search: key-order, params, except=("x")
No-Vary-Search: params=?0	(omit the header)
No-Vary-Search: params=()	(omit the header)
No-Vary-Search: key-order=?0	(omit the header)

Table 2

5.3. Parse a key

To _parse a key_ given an ASCII string _keyString_:

1. Let _keyBytes_ be the isomorphic encoding (https://infra.spec.whatwg.org/#isomorphic-encode) [WHATWG-INFRA] of _keyString_.

2. Replace any 0x2B (+) in _keyBytes_ with 0x20 (SP).

3. Let _keyBytesDecoded_ be the percent-decoding (https://url.spec.whatwg.org/#percent-decode) [WHATWG-URL] of _keyBytes_.

4. Let _keyStringDecoded_ be the UTF-8 decoding without BOM (https://encoding.spec.whatwg.org/#utf-8-decode-without-bom) [WHATWG-ENCODING] of _keyBytesDecoded_.

5. Return _keyStringDecoded_.

5.3.1. Examples

The parse a key algorithm allows encoding non-ASCII key strings in the ASCII structured header format, similar to how the application/x-www-form-urlencoded (https://url.spec.whatwg.org/#concept-urlencoded) format [WHATWG-URL] allows encoding an entire entry list of keys and values in ASCII URL format. For example,

```
No-Vary-Search: params=("%C3%A9+%E6%B0%97")
```

will result in a URL search variance whose vary params are 束 "辿 羂 . As explained in a later example, the canonicalization process during equivalence testing means this will treat as equivalent URL strings such as:

* https://example.com/? 気=1

* https://example.com/?+気=2

* https://example.com/?%C3%A9%20気=3

* https://example.com/?%C3%A9+%E6%B0%97=4

and so on, since they all are parsed (https://url.spec.whatwg.org/#concept-urlencoded-parser) [WHATWG-URL] to having the same key " 気".

6. Comparing

Two URLs (https://url.spec.whatwg.org/#concept-url) [WHATWG-URL] _urlA_ and _urlB_ are _equivalent modulo search variance_ given a URL search variance _searchVariance_ if the following algorithm returns true:

1. If the scheme, username, password, host, port, or path of _urlA_ and _urlB_ differ, then return false.

2. If _searchVariance_ is equivalent to the default URL search variance, then:

    1. If _urlA_’s query equals _urlB_’s query, then return true.

2. Return false.

In this case, even URL pairs that might appear the same after running the application/x-www-form-urlencoded parser (https://url.spec.whatwg.org/#concept-urlencoded-parser) [WHATWG-URL] on their queries, such as https://example.com/a and https://example.com/a?, or https://example.com/foo?a=b&&&c and https://example.com/foo?a=b&c=, will be treated as inequivalent.

3. Let _searchParamsA_ and _searchParamsB_ be empty lists.

4. If _urlA_'s query is not null, then set _searchParamsA_ to the result of running the application/x-www-form-urlencoded parser (https://url.spec.whatwg.org/#concept-urlencoded-parser) [WHATWG-URL] given the isomorphic encoding (https://infra.spec.whatwg.org/#isomorphic-encode) [WHATWG-INFRA] of _urlA_'s query.

5. If _urlB_'s query is not null, then set _searchParamsB_ to the result of running the application/x-www-form-urlencoded parser (https://url.spec.whatwg.org/#concept-urlencoded-parser) [WHATWG-URL] given the isomorphic encoding (https://infra.spec.whatwg.org/#isomorphic-encode) [WHATWG-INFRA] of _urlB_'s query.

6. If _searchVariance_'s no-vary params is a list, then:

    1. Set _searchParamsA_ to a list containing those items _pair_ in _searchParamsA_ where _searchVariance_'s no-vary params does not contain _pair_[0].

    2. Set _searchParamsB_ to a list containing those items _pair_ in _searchParamsB_ where _searchVariance_'s no-vary params does not contain _pair_[0].

7. Otherwise, if _searchVariance_'s vary params is a list, then:

    1. Set _searchParamsA_ to a list containing those items _pair_ in _searchParamsA_ where _searchVariance_'s vary params contains _pair_[0].

    2. Set _searchParamsB_ to a list containing those items _pair_ in _searchParamsB_ where _searchVariance_'s vary params contains _pair_[0].

8. If _searchVariance_'s vary on key order is false, then:

1. Let _keyLessThan_ be an algorithm taking as inputs two pairs (_keyA_, _valueA_) and (_keyB_, _valueB_), which returns whether _keyA_ is code unit less than (https://infra.spec.whatwg.org/#code-unit-less-than) [WHATWG-INFRA] _keyB_.

2. Set _searchParamsA_ to the result of sorting _searchParamsA_ in ascending order with _keyLessThan_.

3. Set _searchParamsB_ to the result of sorting _searchParamsB_ in ascending order with _keyLessThan_.

9. If _searchParamsA_'s size is not equal to _searchParamsB_'s size, then return false.

10. Let _i_ be 0.

11. While _i_ < _searchParamsA_'s size:

    1. If _searchParamsA_[_i_][0] does not equal _searchParamsB_[_i_][0], then return false.

    2. If _searchParamsA_[_i_][1] does not equal _searchParamsB_[_i_][1], then return false.

    3. Set _i_ to _i_ + 1.

12. Return true.

6.1. Examples

Due to how the application/x-www-form-urlencoded parser canonicalizes query strings, there are some cases where query strings which do not appear obviously equivalent, will end up being treated as equivalent after parsing.

So, for example, given any non-default value for No-Vary-Search, such as No-Vary-Search: key-order, we will have the following equivalences:

https://example.com
https://example.com/?
   A null query is parsed the same as an empty string

https://example.com/?a=x
https://example.com/?%61=%78
   Parsing performs percent-decoding

```
https://example.com/?a=
https://example.com/?a=%C3%A9
   Parsing performs percent-decoding

https://example.com/?a=%f6
https://example.com/?a=%ef%bf%bd
   Both values are parsed as U+FFFD ()

https://example.com/?a=x&&&&
https://example.com/?a=x
   Parsing splits on & and discards empty strings

https://example.com/?a=
https://example.com/?a
   Both parse as having an empty string value for a

https://example.com/?a=%20
https://example.com/?a=+
https://example.com/?a= &
   + and %20 are both parsed as U+0020 SPACE
```

7. Caching

If a cache [HTTP-CACHING] implements this specification, the presented target URI requirement in Section 4 of [HTTP-CACHING] is replaced with:

* one of the following:

  - the presented target URI (Section 7.1 of [HTTP]) and that of the stored response match, or

  - the presented target URI and that of the stored response are equivalent modulo search variance (Section 6), given the variance obtained (Section 5.2) from the stored response.

Cache implementations MAY fail to reuse a stored response whose target URI matches _only_ modulo URL search variance, if the cache has more recently stored a response which:

* has a target URI which is equal to the presented target URI, excluding the query, and

* has a non-empty value for the No-Vary-Search field, and

* has a No-Vary-Search field value different from the stored response being considered for reuse.

> Caches aren’t required to reuse stored responses, generally. However, the above expressly empowers caches to, if it is advantageous for performance or other reasons, search a smaller number of stored responses.
>
> That is, because caches might store more than one response for a given pathname, they need a way to efficiently look up the No-Vary-Search value without accessing all cached responses. Such a cache might take steps like the following to identify a stored response in a performant way, before checking the other conditions in Section 4 of [HTTP-CACHING]:
>
> 1. Let exactMatch be cache[presentedTargetURI]. If it is a stored response that can be reused, return it.
>
> 2. Let targetPath be presentedTargetURI, with query parameters removed.
>
> 3. Let lastNVS be mostRecentNVS[targetPath]. If it does not exist, return null.
>
> 4. Let simplifiedURL be the result of simplifying presentedTargetURI according to lastNVS (by removing query parameters which are not significant, and stable sorting parameters by key, if key order is to be be ignored).
>
> 5. Let nvsMatch be cache[simplifiedURL]. If it does not exist, return null. (It is assumed that this was written when storing in the cache, in addition to the exact URL.)
>
> 6. Let searchVariance be obtained (Section 5.2) from nvsMatch.
>
> 7. If nvsMatch’s target URI and presentedTargetURI are not equivalent modulo search variance (Section 6) given searchVariance, then return null.
>
> 8. If nvsMatch is a stored response that can be reused, return it. Otherwise, return null.

To aid cache implementation efficiency, servers SHOULD NOT send different non-empty values for the No-Vary-Search field in response to requests for a given pathname over time, unless there is a need to update how they handle the query component. Doing so would cause cache implementations that use a strategy like the above to miss some stored responses that could otherwise have been reused.

8. Security Considerations

The main risk to be aware of is the impact of mismatched URLs. In particular, this could cause the user to see a response that was originally fetched from a URL different from the one displayed when they hovered a link, or the URL displayed in the URL bar.

However, since the impact is limited to query parameters, this does not cross the relevant security boundary, which is the origin (https://html.spec.whatwg.org/multipage/browsers.html#concept-origin) [HTML]. (Or perhaps just the host (https://url.spec.whatwg.org/#concept-url-host), from the perspective of web browser security UI (https://url.spec.whatwg.org/#url-rendering-simplification). [WHATWG-URL]) Indeed, we have already given origins complete control over how they present the (URL, reponse body) pair, including on the client side via technology such as history.replaceState() (https://html.spec.whatwg.org/multipage/nav-history-apis.html#dom-history-replacestate) or service workers.

9. Privacy Considerations

This proposal is adjacent to the highly-privacy-relevant space of navigational tracking (https://privacycg.github.io/nav-tracking-mitigations/#terminology), which often uses query parameters to pass along user identifiers. However, we believe this proposal itself does not have privacy impacts. It does not interfere with existing navigational tracking mitigations (https://privacycg.github.io/nav-tracking-mitigations/#deployed-mitigations), or any known future ones being contemplated. Indeed, if a page were to encode user identifiers in its URL, the only ability this proposal gives is to _reduce_ such user tracking by preventing server processing of such user IDs (since the server is bypassed in favor of the cache). [NAV-TRACKING-MITIGATIONS]

10. IANA Considerations

IANA should do the following:

10.1. HTTP Field Names

Enter the following into the Hypertext Transfer Protocol (HTTP) Field Name Registry:

Field Name No-Vary-Search

Status permanent

Structured Type Dictionary

Reference this document

Comments (none)

11. References

11.1. Normative References

[FETCH] van Kesteren, A., "Fetch Living Standard", n.d., <https://fetch.spec.whatwg.org/>. WHATWG

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[HTTP-CACHING] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Caching", STD 98, RFC 9111, DOI 10.17487/RFC9111, June 2022, <https://www.rfc-editor.org/rfc/rfc9111>.

[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/rfc/rfc2119>.

[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/rfc/rfc8174>.

[STRUCTURED-FIELDS] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

[WHATWG-ENCODING] van Kesteren, A., "Encoding Living Standard", n.d., <https://encoding.spec.whatwg.org/>. WHATWG

[WHATWG-INFRA] van Kesteren, A. and D. Denicola, "Infra Living Standard", n.d., <https://infra.spec.whatwg.org/>. WHATWG

[WHATWG-URL] van Kesteren, A., "URL Living Standard", n.d., <https://url.spec.whatwg.org/>. WHATWG

11.2. Informative References

[HTML] van Kesteren, A., "HTML Living Standard", n.d., <https://html.spec.whatwg.org/>. WHATWG

[NAV-TRACKING-MITIGATIONS] Snyder, P. and J. Yasskin, "Navigational-Tracking Mitigations", n.d., <https://privacycg.github.io/nav-tracking-mitigations/>. W3C Privacy CG

Acknowledgments

This document benefited from valuable reviews and suggestions by:

* Adam Rice
* Julian Reschke
* Kevin McNee
* Liviu Tinta
* Mark Nottingham
* Martin Thomson
* Valentin Gosu

Index

Authors' Addresses

Domenic Denicola
Google LLC
Email: d@domenic.me

Jeremy Roman
Google LLC
Email: jbroman@chromium.org