Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 16, 2021

K. Dunglas
Les-Tilleuls.coop

September 12, 2020

The Vulcain Protocol
draft-dunglas-vulcain-01

Abstract

This specification defines new HTTP headers (and query parameters) allowing a client to inform the server of the exact data it needs:

* "Preload" informs the server that relations of the main requested resource will be necessary. The server can then reduce the number of round-trips by sending the related resources ahead of time using HTTP/2 [RFC7540] Server Push. When using Server Push isn't possible (resources served by a different authority, client or server not supporting HTTP/2...), the server can hint the client to fetch those resources as early as possible by using the "preload" link relation [W3C.CR-preload-20190626] and the "103" status code [RFC8297].

* "Fields" informs the server of the list of fields of the retrieved resources that will be used. In order to improve performance and reduce bandwidth usage, the server can omit the fields not requested.

Table of Contents

1. Terminology

The keywords MUST, MUST NOT, REQUIRED, SHALL, SHALL NOT, SHOULD, *SHOULD NOT*, RECOMMENDED, MAY, and OPTIONAL, when they appear in this document, are to be interpreted as described in [RFC2119].

2. Preload Header

Many formats including HTML [W3C.REC-html52-20171214], JSON-LD [W3C.REC-json-ld-20140116], Atom [RFC4287], XML [W3C.REC-xml-20081126], HAL (https://tools.ietf.org/html/draft-kelly-json-hal-08) and JSON:API (https://jsonapi.org/) allow the use of Web Linking [RFC5988] to represent references between resources.

The "Preload" HTTP header allows the client to ask the server to transmit resources linked to the main resource it will need as soon as possible.

"Preload" is a List Structured Header [I-D.ietf-httpbis-header-structure]. Its values "MUST" be Strings (Section 3.3.3 of [I-D.ietf-httpbis-header-structure]). Its ABNF is:

```
Preload = sf-list
sf-item = sf-string
```

Its values are selectors Section 4 matching links to resources that "SHOULD" be preloaded. If a value is an empty String, then all links of the current documents are matched.

The server "MUST" recursively follow links matched by the selector. When a selector traverses several resources, all the traversed resources "SHOULD" be sent to the client. If several links referencing the same resource are selected, this resource "MUST" be sent at most once.

The server "MAY" limit the number resources that it sends in response to one request.

Example:

```
Preload: "/member/*/author", "/member/*/comments"
```

The following optional parameters are defined:

* A Parameter whose name is "rel", and whose value is a String (Section 3.3.3 of [I-D.ietf-httpbis-header-structure]) or a Token (Section 3.3.4 of [I-D.ietf-httpbis-header-structure]), conveying the expected relation type of the selected links.

* A Parameter whose name is "hreflang", and whose value is a String (Section 3.3.3 of [I-D.ietf-httpbis-header-structure]), conveying the expected language of the selected links.

* A Parameter whose name is "type", and whose value is a String (Section 3.3.3 of [I-D.ietf-httpbis-header-structure]), conveying the expected media type of the selected links.

The "rel" parameter contains a relation type as defined in [RFC5988]. If this parameter is provided, the server "SHOULD" preload only relations matched by the provided selector and having this type.

The "hreflang" parameter contains a language as defined in [RFC5988]. If this parameter is provided, the server "SHOULD" preload only relations matched by the provided selector and in this language. When possible (for instance, when doing a HTTP/2 Server Push), the server "SHOULD" set the "Accept-Language" request header to this value. If the "hreflang" parameter isn't provided but the server is able to guess the expected language of the relation using other mechanisms (such as the "hreflang" attribute defined by the Atom format for the "atom:link" element, [RFC4287] Section 4.2.7.4), then the "Accept-Language" request header "SHOULD" be set to the guessed value.

The "type" parameter contains a media type as defined in [RFC5988]. If this parameter is provided, the server "SHOULD" preload only relations matched by the provided selector and having this media type. When possible (for instance, when doing a HTTP/2 Server Push), the server "SHOULD" set the "Accept" request header to this value. If the "type" parameter isn't provided but the server is able to guess the expected media type of the relation using other mechanisms (such as the "type" attribute defined by the Atom format for the "atom:link" element, [RFC4287] Section 4.2.7.3), then the "Accept"

request header "SHOULD" be set to the guessed value.

If several parameters are provided for the same selector, the server "SHOULD" preload only relations matching the selector and constraints hinted by the parameters.

Examples:

```
Preload: "/member/*/author"; hreflang="fr-FR"
Preload: "/member/*/author/avatar"; type="image/webp"
```

The server "SHOULD" preload all links matched by the "/member/*/author" selector and having a lang of "fr-FR", as well as all links matching the "/member/*/author/avatar" selector and having a type of "image/webp".

```
Preload: ""; rel=author
Preload: ""; rel="https://example.com/custom-rel"
```


The server "SHOULD" preload all links of the requested resource having the relation type "author" or "https://example.com/custom-rel".

2.1. Preload Example

Considering the following resources:

"/books"

```
{
    "member": [
        "/books/1",
        "/books/2"
    ]
}
```

"/books/1"

```
{
    "title": "1984",
    "author": "/authors/1"
```

```
}

"/books/2"

{
    "title": "Homage to Catalonia",
    "author": "/authors/1"
}

"/authors/1"

{
    "givenName": "George",
    "familyName": "Orwell"
}
```

The "Preload" HTTP header can be used to ask the server to immediately push resources related to the requested one:

```
GET /books/ HTTP/2
Preload: "/member/*/author"
```

In addition to "/books", the server "SHOULD" use HTTP/2 Server Push to push the "/books/1", "/books/2" and "/authors/1" resources. While it is referenced twice, "/authors/1" "MUST" be pushed only once.

Server Push requests generated by the server for related resources "MUST" include the remaining selector in a "Preload" HTTP header. When requesting a pushed relation, the client "MUST" compute the remaining selector and pass it in the "Preload" header.

Explicit Request:

```
GET /books/ HTTP/2
Preload: "/member/*/author"
```

Request to a relation generated by the server (for the push) and the client:

```
GET /books/1 HTTP/2
Preload: "/author"
```

2.2. Using Preload Link Relations

Sometimes, it's not possible or beneficial to use HTTP/2 Server Push: reference to a resource not served by the same authority, client or server not supporting HTTP/2, client having disabled Server Push, resource probably already stored in the cache of the client... To hint the client to preload the resources by initiating and early request, the server "CAN" add references to the resources to preload using "preload" link relations [W3C.CR-preload-20190626].

3. Fields Header

The "Fields" HTTP header allows the client to ask the server to return only the specified fields of the requested resource, and of the preloaded related resources.

The "Fields" HTTP header is a List Structured Header accepting the exact same values than the "Preload" HTTP header defined in Section 2.

The "Fields" HTTP header "MUST" contain a selector (see #Selector). The server "SHOULD" return only the fields matching this selector.

All matched fields "MUST" be returned if they exist. Other fields of the resource "MAY" be omitted.

3.1. Fields Example

Considering the following resources:

"/books/1"


```
{
    "title": "1984",
    "genre": "novel",
    "author": "/authors/1"
}
```

"/authors/1"

```
{
    "givenName": "George",
    "familyName": "Orwell"
}
```

And the following HTTP request:

```
GET /books/1 HTTP/2
Preload: "/author"
Fields: "/author/familyName", "/genre"
```

The server must return a response containing the following JSON document:

```
{
    "genre": "novel",
    "author": "/authors/1"
}
```

And push the following filtered "/authors/1" resource:

```
{
    "familyName": "Orwell"
}
```

Server Push requests generated by the server for related resources "MUST" include the remaining selector in a "Fields" HTTP header. When requesting a pushed relation, the client "MUST" compute the remaining selector and pass it in the "Fields" header.

Example:

Explicit Request:

```
GET /books/ HTTP/2
Fields: "/member/*/author"
```

Request to a relation generated by the server (for the push) and the client:


```
GET /books/1 HTTP/2
```

```
Fields: "/author"
```

4. Selectors

Selectors used as value of the "Preload" and "Fields" HTTP headers depend on the "Content-Type" of the requested resource. This specification defines default selector formats for common content-types, and a mechanism to use other selector formats.

The client "SHOULD" use the "Accept" HTTP header to request the resource in a format compatible with selectors used in "Preload" and "Fields" HTTP headers.

The client can use the "Prefer" HTTP header [RFC7240] with the "selector" preference to ask the server to use a specific selector format:

```
GET /books/1 HTTP/2
Accept: text/xml
Prefer: selector=css
Fields: "brand > name"
```

If no explicit preferences have been passed, the server "MUST" assume that the selector format is the default corresponding to the format of the resource.

The following table defines the default selector format for common formats:

Format	Selector format	Identifier
JSON	Extended JSON Pointer Section 4.1	json-pointer
XML	XPath [W3C.REC-xpath-19991116]	xpath
HTML	CSS selectors [W3C.REC-selectors-3-20181106]	css

Table 1

The client and the server can negotiate the use of other selector formats using the "Prefer" HTTP header.

4.1. Extended JSON Pointer

For JSON documents, the default selector format is JSON Pointer [RFC6901]. However, JSON Pointer doesn't provide a mechanism to select entire collections.

This specification defines an extension to the JSON Pointer format allowing to select every element of a collection, the "*" character.

Considering the following JSON document:

```
{
    "books": [
        {
            "title": "1984",
            "author": "George Orwell"
        },
        {
            "title": "The Handmaid's Tale",
            "author": "Margaret Atwood"
        }
    ]
}
```

The "/books/*/author" JSON Pointer selects the "author" field of every objects in the "books" array.

The "*" character is escaped by encoding it as the "~2" character sequence.

By design, this selector is simple and limited. Simple selectors make it easier to limit the complexity of requests executed by the server.

5. Query Parameters

Another option available to clients is to utilize Request URI query-string parameters to pass preload and fields selectors.

The "preload" and "query" parameters "MAY" be used to pass selectors corresponding respectively to the "Preload" and "Fields" HTTP headers. Valid values for these query parameters are exactly the same than the ones defined of the "Preload" and "Fields" HTTP headers.

In conformance with the Section 3.4 of the URI RFC [RFC3986], values

of query parameters "MUST" be percent-encoded.

For instance, the list of fields selector ""/title","/author"" and the preload selector ""/author"" passed using query parameters will result in the following URL: "/books/1?fields=%22%2Ftitle%22%2C%22%2Fauthor%22&preload=%22%2Fauthor%22".

When using query parameters, the server "MUST" pass the remaining part of the selector as parameter of the generated link.

"Preload" and "Fields" HTTP headers aren't CORS safe-listed request-headers (https://fetch.spec.whatwg.org/#cors-safelisted-request-header). Query parameters, on the other hand, allow to send cross-site requests that don't trigger preflight requests. Also, query parameters don't require clients to compute the remaining part of the selector when requesting relations.

However, support for query parameters can be challenging to implement by servers (links contained in served documents "MUST" be modified) and generate URLs that are hard to read for a human.

Altering the URI can also have undesirable effects.

For these reasons, using HTTP headers "SHOULD" be preferred. Support for query parameters is "OPTIONAL". A server supporting query parameters "MUST" also support the corresponding HTTP headers.

Example:

```
GET /books/?preload=%22%2Fmember%2F%2A%2Fauthor%22 HTTP/2

{
    "member": {
        "/books/1?preload=%22%2Fauthor%22",
        "/books/1?preload=%22%2Fauthor%22"
    }
}
```

Example using parameters:

```
GET /books/?preload=%22%2Fmember%2F%2A%22%3B%20rel%3Dauthor HTTP/2
```

```
{
    "member": {
        "/books/1?preload=%22%22%3B%20rel%3Dauthor",
        "/books/1?preload=%22%22%3B%20rel%3Dauthor"
    }
}
```


6. Computing Links Server-Side

While using hypermedia capabilities of the HTTP protocol through Web Linking "SHOULD" always be preferred, sometimes links between resources are known by the server but are not provided in the HTTP response.

In such cases, the server can compute the link server-side in order to push the related resource. Such server-side computed links "MAY" be documented, for instance by providing an OpenAPI specification (https://www.openapis.org/) containing Link objects (http://spec.openapis.org/oas/v3.0.2#link-object).

Considering the following resources and assuming that the server knows that the "author" field references the resources "/authors/{id}" resource:

"/books/1"

```
{
    "title": "1984",
    "author": 1
}
```

"/authors/1"

```
{
    "givenName": "George",
    "familyName": "Orwell"
}
```

In response to this request , both "/books/1" and "/authors/1" should

be pushed:

```
GET /books/1 HTTP/2
Preload: "/author"
```

7. Security Considerations

Using the "Preload" header can lead to a large number of resources to be generated and pushed. The server "SHOULD" limit the maximum number of resources to push. The depth of the selector "SHOULD" also be limited by the server.

8. IANA considerations

The "Preload"and "Fields" header fields will be added to the "Permanent Message Header Field Names" registry defined in [RFC3864].

A selector registry could also be added.

9. Implementation Status

[RFC Editor Note: Please remove this entire section prior to publication as an RFC.]

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist. According to RFC 6982, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit."

9.1. Vulcain Gateway Server

Organization responsible for the implementation:

Les-Tilleuls.coop

Implementation Name and Details:

Vulcain.rocks, available at https://vulcain.rocks (https://vulcain.rocks)

Brief Description:

A gateway server allowing to add support for the Vulcain protocol to any existing API. It is written in Go and is optimized for performance.

Level of Maturity:

Beta.

Coverage:

All the features of the protocol as well as the extended JSON pointer selector.

Version compatibility:

The implementation follows the draft version 00.

Licensing:

All code is covered under the GNU Affero Public License version 3 or later.

Implementation Experience:

Used in production.

Contact Information:

Kévin Dunglas, kevin+vulcain@dunglas.fr (mailto:kevin+vulcain@dunglas.fr) https://vulcain.rocks (https://vulcain.rocks)

Interoperability:

Reported compatible with all major browsers and server-side tools.

9.2. Helix Vulcain Filters

Organization responsible for the implementation:

Adobe

Implementation Name and Details:

Helix Vulcain Filters, available at https://github.com/adobe/helix-vulcain-filters (https://github.com/adobe/helix-vulcain-filters)

Brief Description:

Vulcain-like filters for OpenWhisk web actions.

Level of Maturity:

Stable.

Coverage:

HTTP headers as well as the extended JSON pointer selector.

Version compatibility:

The implementation follows the draft version 00.

Licensing:

All code is covered under the Apache License 2.0.

Implementation Experience:

Used in production.

Contact Information:

https://www.adobe.com/about-adobe/contact.html
(https://www.adobe.com/about-adobe/contact.html)

Interoperability:

Reported compatible with all major browsers and server-side tools.

10. Acknowledgements

The author would like to thank Evert Pot, who authored the Prefer-Push Internet-Draft from which some parts of this specification is inspired, and André R. who gave good design ideas.

11. Normative References

[RFC7240] Snell, J., "Prefer Header for HTTP", RFC 7240, DOI 10.17487/RFC7240, June 2014, <https://www.rfc-editor.org/info/rfc7240>.

[W3C.REC-selectors-3-20181106] A elik, T., Etemad, E., Glazman, D., Hickson, I., Linss, P., and J. Williams, "Selectors Level 3", World Wide Web Consortium Recommendation REC-selectors-3-20181106, November 6, 2018, <https://www.w3.org/TR/2018/REC-selectors-3-20181106>.

[RFC6901] Bryan, P., Ed., Zyp, K., and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Pointer", RFC 6901, DOI 10.17487/RFC6901, April 2013, <https://www.rfc-editor.org/info/rfc6901>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <https://www.rfc-editor.org/info/rfc7540>.

[W3C.REC-xpath-19991116] Clark, J. and S. DeRose, "XML Path Language (XPath) Version 1.0", World Wide Web Consortium Recommendation REC-xpath-19991116, November 16, 1999, <https://www.w3.org/TR/1999/REC-xpath-19991116>.

[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>.

[RFC8297] Oku, K., "An HTTP Status Code for Indicating Hints", RFC 8297, DOI 10.17487/RFC8297, December 2017, <https://www.rfc-editor.org/info/rfc8297>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <https://www.rfc-editor.org/info/rfc5988>.

[I-D.ietf-httpbis-header-structure] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", Work in Progress, Internet-Draft, draft-ietf-httpbis-header-structure-19, June 3, 2020, <https://tools.ietf.org/html/draft-ietf-httpbis-header-structure-19>.

[W3C.CR-preload-20190626] Grigorik, I. and Y. Weiss, "Preload", World Wide Web Consortium CR CR-preload-20190626, June 26, 2019, <https://www.w3.org/TR/2019/CR-preload-20190626>.

[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>.

12. Informative References

[W3C.REC-html52-20171214]
Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, December 14, 2017, <https://www.w3.org/TR/2017/REC-html52-20171214>.

[W3C.REC-xml-20081126]
Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 26, 2008, <https://www.w3.org/TR/2008/REC-xml-20081126>.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, DOI 10.17487/RFC4287, December 2005, <https://www.rfc-editor.org/info/rfc4287>.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, DOI 10.17487/RFC6982, July 2013, <https://www.rfc-editor.org/info/rfc6982>.

[W3C.REC-json-ld-20140116]
Sporny, M., Kellogg, G., and M. Lanthaler, "JSON-LD 1.0", World Wide Web Consortium Recommendation REC-json-ld-20140116, January 16, 2014, <https://www.w3.org/TR/2014/REC-json-ld-20140116>.

Author's Address

Kevin Dunglas
Les-Tilleuls.coop
82 rue Winston Churchill
59160 Lille
France

Email: kevin@les-tilleuls.coop