HTTPbis
Internet-Draft
Intended status: Experimental
Expires: September 6, 2019

M. Thomson
Mozilla
March 05, 2019

Identifying HTTP Exchanges with URIs
draft-thomson-http-hx-uri-00

Abstract

URI schemes are defined that enable identification of HTTP exchanges, or parts of those exchanges.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

It is common for applications that use HTTP [HTTP] to use a "follow your nose" design. In this design, clients make requests to discover or create resources and to learn information about resources they are interested in. Once the identity of resources is learned, clients then interact with those resources. This process is often iterative, with clients following multiple links to reach resources of interest.

A negative consequence of these designs is that the discovery or creation steps add latency to any operation that depends on the identity of resources.

For applications that use well-defined formats, though the result of a request might be unknown, an application might have reliable knowledge about the form of the response. If components of that answer could be incorporated into another request by reference, then the application might save a round trip for every such occurrence.

The "hx" URI scheme identifies components of HTTP exchanges. The "hxr" URI scheme provides for further indirection, allowing the dereferencing of URLs in identified HTTP exchanges.

1.1. Example

In this simple example, a client wishes to create and then update a resource.

```
POST /make-object?name=example HTTP/1.1
Host: example.com
```

The server creates a resource and provides its location in a response:

```
HTTP/1.1 201 Created
Location: https://example.com/roZ2ITW
Content-Type: example/example+json

{
  "uri": "https://example.com/roZ2ITW",
  "name": "example",
  "items": { "a": 1, "b": 2 }
}
```

After receiving the identity of the resource, the client can then interact with that resource, here copying the value of "b" to a new key called "c":

```
POST /roZ2ITW HTTP/1.1
Host: example.com

add_item: c=2
```

With an "hx" URI, and support from the server, the client can send the second request at the same time as the first, relying on the server to dereference the "hxr" URI:

```
POST hxr:///0/a/h/location?201 HTTP/1.1
Host: example.com

add_item: c=@hx:///0/a/b?ct=example%2fexample+json#/items/b
```

If the server understands the "hxr" URI scheme, it dereferences that URI to determine the target of the request. The value from /items/b (using JSON Pointer [RFC6901]) is copied using the "hx" URI scheme.

Note that it can be seen that though the initial POST request that creates the resource is not idempotent, the client is able to construct the next request in a way that ensures that they are conditional on the outcome of that request. This ensures that the transaction does not complete unless all requests are successful.

1.2. 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.

1.3. Terminology

This document uses terminology from HTTP [HTTP]. The phrase HTTP URI is used to refer to http:// and https:// URIs collectively. However, this document is only capable of identifying requests that are sent over secured transports.

2. Overview

An "hx" URI identifies an HTTP exchange, or parts of that exchange. The primary advantage of this in referring to the product of a request before it is completed.

A scheme of "hx" is followed by an authority that identifies the connection on which the exchange was initiated. A minimal path includes an identifier for the exchange, as a decimal number. For

instance, assuming that the authority "b5dd5901aef3f33de572" refers to an HTTP/2 connection, the following URI identifies entire exchange on stream 7 of that connection (see Section 4).

```
hx://b5dd5901aef3f33de572/7
```

Adding additional path elements narrows this to refer to the request (see Section 5):

```
hx://b5dd5901aef3f33de572/7/q
```

Further path elements allow components of a message to be identified, such as a Location header field value (see Section 6.8):

```
hx://b5dd5901aef3f33de572/7/a/h/location
```

To ensure that the Location header field is only used if the request resulted in the creation of a new resource (that is, the response had a 201 (Created) status code), conditions can be added to the URI as query parameters:

```
hx://b5dd5901aef3f33de572/7/a/h/location?201
```

A fragment can be used if the content has an associated content type that supports fragment identifiers, which is generally only possible for the body of a request or response:

```
hx://b5dd5901aef3f33de572/7/a/b#title
```

How a fragment is used depends on the content type of the identified resource, so a condition might be added to specify the content type of the target resource:

```
hx://b5dd5901aef3f33de572/7/a/b?ct=text%2Fhtml#title
```

The "hxr" URI scheme is identical to "hx" except that it is dereferenced twice. A reference that uses the "hxr" scheme can therefore be used where a URI would otherwise be used, taking the value of the URI from chosen part of the identified exchange.

3. Authority

The authority component of an "hx" or "hxr" URI is an identifier for a connection.

The process for generating a unique identifier uses TLS exporters (see Section 7.5 of [TLS13]). Consequently, exchanges on connections that do not use TLS cannot be identified using "hx" or "hxr" URIs.

A TLS exporter with the label "EXPORTER-hx-authority" and an empty context is used to produce a 10 octet value. This value is then encoded in hexadecimal (that is, Base 16 [RFC4648]) to produce a 20 character authority.

The authority can be omitted where the identity of the connection can be inferred from context. For instance, where the URI is sent over the same connection. The current connection is used if an authority is absent.

```
hx:///7
```

The userinfo and port components of an "hx" or "hxr" URI MUST NOT be used. Any URI with userinfo or port components is invalid.

4. Identifying an Exchange

After identifying the connection, the first element of the path after the initial slash ("/") identifies a request-response exchange.

A decimal value is used to identify an exchange. How requests are identified depends on the version of HTTP in use.

A single "p" character followed by a decimal value is used to identify a server push, see Section 4.4.

An "hx" or "hxr" URI always includes fields that identify an exchange. For instance, the following URIs are incomplete and therefore invalid:

```
hx://
hx:///
hx://b5dd5901aef3f33de572/
```

4.1. Identifying HTTP/1.1 Exchanges

In HTTP/1.1 [HTTP11] and earlier, the numeric identifier for an exchange counts the number of exchanges on the connection that precede the target exchange. The first exchange on a connection is therefore identified as "hx:///0". Subsequent requests increment this value by 1.

4.2. Identifying HTTP/2 Exchanges

In HTTP/2 [HTTP2], the numeric identifier for an exchange corresponds to a HTTP/2 stream identifier. The first exchange on a connection is therefore identified as "hx:///1". As a result, all exchanges that are not server pushes use odd-numbered identifiers.

4.3. Identifiers HTTP/3 Exchanges (#exchange3}

In HTTP/3 [HTTP3], the numeric identifier for an exchange corresponds to a QUIC stream identifier. The first exchange on a connection is therefore identified as "hx:///0". Consequently, all exchanges that are not server pushes use identifiers that are whole multiples of 4.

4.4. Identifying Server Pushes

A server push exchange is identified by a "p" prefix followed by a decimal value. For example:

```
hx:///p6
```

In HTTP/2, a stream identifier is sufficient to distinguish between requests and server pushes. Thus, identifying a server push is possible even if the "p" prefix is omitted. In HTTP/2, all server pushes use even-numbered identifiers.

Server pushes in HTTP/3 are given a push ID, an identifier that might be the same as the stream ID used for requests. The push ID is used to identify server push in HTTP/3. Thus, in HTTP/3, the "p" prefix is necessary to properly identify a server push.

HTTP versions prior to HTTP/2 do not provide server push, so an "hx" URI that attempts to identify a server push cannot be successfully resolved.

5. Targets

Without further qualification, an "hx" URI identifies a message exchange, both the request and the response as a whole. Applications can narrow this to a request (Section 5.1) or response (Section 5.2).

A "hxr" URI always identifies a request or response, it cannot identify a complete exchange.

5.1. Identifying a Request

A request is identified by adding "/q" to a URI identifying an exchange. For example:

```
hx://546c9bce274b06cf859d/84/q
```

5.2. Identifying a Response

A request is identified by adding "/a" to a URI identifying an exchange. For example:

```
hx://18660225619af2c6c300/173/a
```

5.3. Redirections

An "hx" or "hxr" URI applies to a single exchange over a single connection. If a 3xx status code results in a client following a redirect, that exchange is identified separately.

6. Identifying Request or Response Components

After identifying a single message, additional path components can be used to identify parts of the message.

TBD: It might make sense to put "/m", "/u", "/s", and "/i" as peers to "/q" and "/a" rather than attaching them underneath. The primary advantage would be a shorter identifier. (Doing this for "/i" alone might work, as that is more of a peer to "/a".)

6.1. Identifying the Request Method

A path component of "/m" indicates that an 'hx' URI identifies the request method. This component is not valid for an "hxr" URI or an "hx" URI that identifies a response.

6.2. Identifying the Effective Request URI

A path component of "/u" indicates that an 'hx' or "hxr" URI identifies the effective request URI (see Section 5.3 of [HTTP]).

6.3. Identifying the Response Status

A path component of "/s" indicates that an 'hx' URI identifies the request method. This component is not valid for an "hxr" URI or an "hx" URI for a request.

6.4. Identifying the Message Body

A path component of "/b" identifies the body of the message. A body can be identified for both "hx" and "hxr" URIs.

Identifying the body of a message without a body (like a GET request or a 204 (No Content) response) successfully identifies the empty body.

6.5. Informational (1xx) Responses

The "/i" path component can be used to select informational responses.

The "/i" component is followed by a path component that identifies which informational responses to select. If this contains a decimal value, this indicates the number of the informational response to select. The first information response is identified with "0", with subsequent information responses each using a number 1 greater than the last. A value of "@" is used to identify the last informational response and a value of "*" identifies all informational responses.

TBD: Indexing is a little strange given the use case here. The problem lies in working out what to do with multiple entries. Maybe the right answer is to allow for selecting just the first, last, or all items. That would simplify the scheme a little.

Indexing applies after any conditions are applied (see Section 7), allowing a URI to identify single informational response.

For example, the following "hx" URIs refer to the third 103 response, the last informational response containing a Link header field, and all informational responses respectively.

```
hx:///71/a/i/2?103
hx:///71/a/i/@?h=link
hx:///71/a/i/*
```

The path components "/s" (Section 6.3) or "/h" (Section 6.6) can be used to select parts of an informational response. For example, all Link header fields from informational responses can be collected with:

```
hx:///10/a/i/*/h/link/*
```

6.6. Identifying a Message Header

A path component of "/h" identifies the header of a message as a whole. When preceded by "/i", this identifies the header of the informational response (see Section 6.5). When not preceded by "/i" it refers to the header from requests and final responses.

Without additional path elements, this form is only valid for an "hx" URI; an "hxr" URI requires that specific header fields be identified.

6.7. Identifying a Message Trailer

A path component of "/t" specifically identifies the trailer of a message. Trailers are subject to the same restrictions as headers with the additional condition that they can't be present on informational responses.

6.8. Identifying Header Field Values

Adding a path component containing the name of a header field to a path that identifies the a header ("/h" or "/i/.../h") or trailer ("/t") from a message selects that header field only.

The next path component indexes header field values, just like informational responses are indexed (see Section 6.5). All values from the message are identified by "*". A decimal value indicates a 0-based index into values. The last value is identified by "@".

Values that use the HTTP list construction are not indexed by instances of the header field, but by the comma-separated values that are present. Empty values or those containing only whitespace are skipped and cannot be indexed.

To illustrate this, there are 4 values that can be indexed in the following HTTP/1.1 example. The third value is "3" and the last ("/@") is "4".

```
Example: 1
Example: 2, ,3
Example: ,4,
```

As a special case, an "hxr" URI that refers to the value of a Link header field [LINK] can be used as a reference.

7. Conditions

The query string of an "hx" URI carries a set of conditions. Unless any conditions evaluate to true, the resolution of the URI will fail. This allows for specification of URIs that are conditional on details of the HTTP exchange.

For example, the following URI cannot be dereferenced unless the response indicates success, ensuring that the body of an unwanted response like 503 is not used:

```
hx://b5dd5901aef3f33de572/7/a/b?2xx
```

Conditions are separated by the ampersand ("&") character. Each comprises a label that identifies the type of the condition, and an optional value. The value is separated from the label by an equals sign ("=") character.

This document defines conditions for status code (Section 7.3), header field values (Section 7.4), and response content-type (Section 7.5). Conditions that are not understood always evaluate to false, causing resolution to fail.

7.1. Condition Processing Model

Conditions are potentially processed multiple times.

Multiple values can be produced for informational responses and header fields. In each case, when multiple values are produced, conditions are evaluated. This might reduce the number of options. If multiple values remain, all options are considered when evaluating the remainder of the URI path.

For instance, a Link header field [LINK] might appear multiple times and in multiple informational responses. The Link Relation Type condition (Section 7.6) might be used to select all link relations of a given type across all informational responses.

```
hx:///29/a/i/*/h/link/*?rel=start
```

7.2. Percent-Encoding of Condition Values

The URI grammar [URI] prohibits the use of certain characters in the query string. This scheme uses percent-encoding to allow conditions to carry values that are not permitted by the URI grammar. Section 2.1 of [URI] defines percent-encoding. The "hx-pct-encoded" rule in Section 8 defines the characters that don't require encoding; all other values MUST be percent-encoded.

7.3. Status Condition

Any condition that starts with a numeral from "1" to "5" is used to specify a condition on the response status.

If the condition contains three digits, the condition evaluates to true if the response contained a matching status code.

A condition that contains a numeral and two "x" characters evaluates to true if the status code is from the identified class. For instance, the following identifies a request that was redirected:

```
hx:///22/q?3xx
```

A condition that specifies an informational status code (1xx) will be true if an informational response of that type was present. It does not result in limiting the components that can be selected. Specific 100-series status codes can be used to limit which informational responses are selected if the "/i" path component is used (see Section 6.5).

A URI that identifies a header field will resolve the final value of the header field unless a specific portion of the response is specified (using "/i" or "/t"), taking into account values from final responses and trailers as defined in Section 6.6.

This condition can be used to identify components of a request, conditional on the status code of the response.

New condition definitions MUST NOT start with a numeral from "1" and "5".

7.4. Header Field Value Condition

The header field condition is identified with a "h" token. A "h=" is followed by the name of the header field. With no further values, this condition is satisfied if a header field with the same name is present in the identified part of the message. An additional "=" character can be added, which causes the condition to be true only when the value of the header field is equal to the remainder of the condition.

This condition applies to any header field from the identified object. Thus, if the URI does not specify whether a request or response, the condition is met based on the presence or value of the header or trailer field in request or response, including informational responses. If the target of the URI is a request, response, or informational response, then only header and trailer fields in the corresponding part of the message apply. For instance, if the URI identies the header, then only header fields are used to match.

Thus, to identify a request if it contains a User-Agent header field with any value, the following might be used:

```
hx:///30/q?h=user-agent
```

To select a response body only if it indicates that requests for byte ranges are supported, the following might be used:

```
hx:///71/a/b?h=accept-ranges=bytes
```

Alternative forms of matching aside from equality might be provided in future.

7.5. Response Content Type Condition

The response content type condition matches if the content type of the response matches the specified content type. Acceptable values and rules for determining what values match follow the rules for the Accept header field (see Section 8.4.2 of [HTTP]).

The response content type condition is identified by "ct" and is followed by a percent-encoded content type. For example:

```
hx:///12/a/b?ct=text%2Fhtml
```

Unlike the header field condition (Section 7.4), the response content type condition can be used with URIs that identify components of a request. In that case, it indicates that the identification is conditional on the content type of the response (not the request).

Separator characters ("/", ";" and ",") MUST be percent-encoded in the value of this condition.

7.6. Link Relation Condition

A link relation condition filters results by those that contain a link relation [LINK] of the specified type.

The link relation condition is identified by "rel" and is followed by a link relation type. Link relations that include non-token characters, such as those that use the URL form, MUST be percent-encoded.

If the target is a request, response, informational response, or component that contains header fields, only those messages or parts of messages that contain a link relation of the specified type are selected.

If the target is a Link header field, then only link relations of the identified type are selected. Deferencing fails if any other header field is identified.

8. hx URI Grammar

In ABNF [RFC5234], the "hx" URI scheme can be described as a narrow profile of that defined in [URI].

```
hx-URI = "hx://" [ hx-authority ] hx-exchange
         [ hx-target ] [ hx-conditions ]
hx-authority = 20HEXDIG
hx-exchange = "/" [ "p" ] 1*DIGIT

hx-target = hx-request / hx-response
hx-request = "/q" [ "/" hx-component ]
hx-response = "/a" [ "/" hx-component ]

hx-component = hx-method / hx-uri / hx-info
             / hx-header / hx-body / hx-trailer
hx-method = "/m"
hx-uri = "/u"
hx-info = "/i/" hx-index [ hx-status / hx-header ]
hx-status = "/s"
hx-header = "/h" [ "/" hx-token [ "/" hx-index ] ]
hx-body = "/b"
hx-trailer = "/t" [ "/" hx-token [ "/" hx-index ] ]

hx-index = 1*DIGIT / "@" / "*"
hx-token = 1*hx-token-char
hx-token-char = "-" / "." / "_" / DIGIT / ALPHA

hx-conditions = "?" hx-condition *("&" hx-condition)
hx-condition = hx-status-cond / hx-header-cond
             / hx-ct-cond / hx-extension-cond
hx-status-cond = ("1" / "2" / "3" / "4" / "5") (2DIGIT / "xx")
hx-header-cond = "h=" hx-token [ "=" hx-pct-encoded ]
hx-ct-cond = "ct=" hx-pct-encoded
hx-extension-cond = hx-token [ "=" hx-pct-encoded ]

hx-pct-encoded = *( hx-token-char / ("%" 2HEXDIG) )
```

9. hxr URI Grammar

The "hxr" URI scheme uses the same basic grammar as the "hx" URI scheme. However, since this can only ever reference parts of an exchange that could contain a URI, the grammar is more narrowly defined.

```
   hxr-URI = "hxr://" [ hx-authority ] "/" hx-exchange
             hxr-target [ "?" hx-conditions ]

   hxr-targets = hxr-request / hxr-response
   hxr-request = "/q/" hxr-component
   hxr-response = "/a/" hxr-component

   hxr-component = hx-uri / hxr-info
                 / hxr-header / hx-body / hxr-trailer
   hxr-info = "/i/" hx-index hxr-header
   hxr-header = "/h/" hx-token [ "/" hx-index ]
   hxr-trailer = "/t/" hx-token [ "/" hx-index ]
```

The main difference between the "hx" and "hxr" schemes is that "hxr" URIs contain a narrower set of possible values, omitting all means of identifying parts of a request that cannot produce a URI.

10. Security Considerations

Resolution of details of unfulfilled requests could present a significant state commitment on servers. Servers that receive requests that depend on other requests might have to block processing until the outcome of the referenced requests is complete. Alternatively, servers might need to hold information about completed requests in anticipation of receiving references to that request.

Servers can fail resolution of "hx" or "hxr" URIs if the state required would present an undue burden on their operation. Servers might limit the types of information that can be retained and referenced to reduce this cost.

Applications that use these URI schemes MUST define what types of reference a server is expected to be able to handle, or provide a means of negotiating what can be relied on.

11. IANA Considerations

This document registers the "hx" and "hxr" URI schemes. In support of this, registrations are made in the TLS exporters registry (Section 11.3) and registries are established for managing the parameters of the URI schemes (Section 11.4).

11.1. hx URI scheme Registration

The "hx" URI scheme is registered according to the procedures in [BCP35].

Scheme name: hx

Status: Permanent/Provisional

Applications/protocols that use this scheme name: Applications use URIs with this scheme to identify HTTP exchanges, requests, responses, or components of those messages.

Contact: IETF Chair chair@ietf.org [1]

Change controller: IESG iesg@ietf.org [2]

Reference: This document.

11.2. hxr URI scheme Registration

The "hxr" URI scheme is registered according to the procedures in [BCP35].

Scheme name: hxr

Status: Permanent/Provisional

Applications/protocols that use this scheme name: Applications use URIs with this scheme in place of URIs where the intended URI is found in a component of an HTTP exchange.

Contact: IETF Chair chair@ietf.org [3]

Change controller: IESG iesg@ietf.org [4]

Reference: This document.

11.3. TLS Exporter Registration

TODO

11.4. hx and hxr URI Scheme Registries

TODO considering setting up registries for various bits of the syntax.

12. References

12.1. Normative References

[BCP35] Thaler, D., Ed., Hansen, T., and T. Hardie, "Guidelines and Registration Procedures for URI Schemes", BCP 35, RFC 7595, DOI 10.17487/RFC7595, June 2015, <https://www.rfc-editor.org/info/rfc7595>.

[HTTP] Fielding, R., Nottingham, M., and J. Reschke, "HTTP Semantics", draft-ietf-httpbis-semantics-03 (work in progress), October 2018.

[HTTP11] Fielding, R., Nottingham, M., and J. Reschke, "HTTP/1.1 Messaging", draft-ietf-httpbis-messaging-03 (work in progress), October 2018.

[HTTP2] 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>.

[HTTP3] Bishop, M., "Hypertext Transfer Protocol Version 3 (HTTP/3)", draft-ietf-quic-http-18 (work in progress), January 2019.

[LINK] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <https://www.rfc-editor.org/info/rfc8288>.

[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>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[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>.

[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>.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[URI] 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>.

12.2. Informative References

[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>.

12.3. URIs

[1] mailto:chair@ietf.org

[2] mailto:iesg@ietf.org

[3] mailto:chair@ietf.org

[4] mailto:iesg@ietf.org

Acknowledgments

TODO acknowledge.

Author's Address

Martin Thomson
Mozilla

Email: mt@lowentropy.net