HTTP
Internet-Draft
Intended status: Standards Track
Expires: 2 April 2026

J. Reschke
greenbytes
J.M. Snell
Cloudflare
M. Bishop
Akamai

29 September 2025

The HTTP QUERY Method
draft-ietf-httpbis-safe-method-w-body-12

Abstract

This specification defines the QUERY method for HTTP. A QUERY requests that the request target process the enclosed content in a safe/idempotent manner and then respond with the result of that processing. This is similar to POST requests but can be automatically repeated or restarted without concern for partial state changes.

Editorial Note

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

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/.

Working Group information can be found at https://httpwg.org/; source code and issues list for this draft can be found at https://github.com/httpwg/http-extensions/labels/query-method.

The changes in this draft are summarized in Appendix B.12.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

This specification defines the HTTP QUERY request method as a means of making a safe, idempotent request (Section 9.2 of [HTTP]) containing content that describes how the request is to be processed by the target resource.

Most often, this is desirable when the data conveyed in a request is too voluminous to be encoded into the request's URI. A common query pattern is:

```
GET /feed?q=foo&limit=10&sort=-published HTTP/1.1
Host: example.org
```

However, when the data conveyed is too voluminous to be encoded in the request's URI, this pattern becomes problematic:

* often size limits are not known ahead of time because a request can pass through many uncoordinated systems (but note that Section 4.1 of [HTTP] recommends senders and recipients to support at least 8000 octets),

* expressing certain kinds of data in the target URI is inefficient because of the overhead of encoding that data into a valid URI,

* request URIs are more likely to be logged than request content, and may also turn up in bookmarks,

* encoding queries directly into the request URI effectively casts every possible combination of query inputs as distinct resources.

As an alternative to using GET, many implementations make use of the HTTP POST method to perform queries, as illustrated in the example below. In this case, the input to the query operation is passed as the request content as opposed to using the request URI's query component.

A typical use of HTTP POST for requesting a query is:

```
POST /feed HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded

q=foo&limit=10&sort=-published
```

This variation, however, suffers from the fact that it is not readily apparent -- absent specific knowledge of the resource and server to which the request is being sent -- that a safe, idempotent query is being performed.

The QUERY method provides a solution that spans the gap between the use of GET and POST, with the example above being expressed as:

```
QUERY /feed HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded

q=foo&limit=10&sort=-published
```

As with POST, the input to the query operation is passed as the content of the request rather than as part of the request URI. Unlike POST, however, the method is explicitly safe and idempotent, allowing functions like caching and automatic retries to operate.

Recognizing the design principle that any important resource ought to be identified by a URI, this specification describes how a server can assign URIs to both the query itself or a specific query result, for later use in a GET request.

Summarizing:

	GET	QUERY	POST
Safe	yes	yes	potentially no
Idempotent	yes	yes	potentially no
URI for query itself	yes (by definition)	optional (Location response field)	no
URI for query result	optional (Content-Location response field)	optional (Content-Location response field)	optional (Content-Location response field)
Cacheable	yes	yes	yes, but only for future GET or HEAD requests
Content (body)	"no defined semantics"	expected (semantics per target resource)	expected (semantics per target resource)

Table 1: Summary of relevant method properties

1.1. Terminology

This document uses terminology defined in Section 3 of [HTTP].

Furthermore, it uses the terms _URI query parameter_ for parameters in the query component of a URI (Section 4.2.2 of [HTTP]) and _query content_ for the request content (Section 6.4 of [HTTP]) of a QUERY request.

1.2. 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.

2. QUERY

The QUERY method is used to initiate a server-side query. Unlike the GET method, which requests a representation of the resource identified by the target URI (as defined by Section 7.1 of [HTTP]), the QUERY method is used to ask the target resource to perform a query operation within the scope of that target resource. The query operation is described by the request content. The origin server determines the scope of the operation based on the target resource.

The content of the request and its media type define the query. Servers MUST fail the request if the Content-Type request field ([HTTP], Section 8.3) is missing or is inconsistent with the request content.

As for all HTTP methods, the target URI's query part takes part in identifying the resource being queried. Whether and how it directly affects the result of the query is specific to the resource and out of scope for this specification.

QUERY requests are safe with regard to the target resource ([HTTP], Section 9.2.1) -- that is, the client does not request or expect any change to the state of the target resource. This does not prevent the server from creating additional HTTP resources through which additional information can be retrieved (see Sections 2.3 and 2.4).

Furthermore, QUERY requests are idempotent ([HTTP], Section 9.2.2) -- they can be retried or repeated when needed, for instance after a connection failure.

As per Section 15.3 of [HTTP], a 2xx (Successful) response code signals that the request was successfully received, understood, and accepted.

In particular, a 200 (OK) response indicates that the query was successfully processed and the results of that processing are enclosed as the response content.

2.1. Media Types and Content Negotiation

The semantics of a QUERY request depends both on the request content and the associated metadata, such as the Media Type ([HTTP], Section 8.3.1). In general, any problem with requests where content and metadata are inconsistent MUST be rejected with a 4xx (Client Error) response ([HTTP], Section 15.5).

The list below describe various cases of failures and recommends specific status codes:

* A request lacking media type information by definition is incorrect and needs to fail with a 4xx status code such as 400 (Client Error).

* If a media type is specified, but not supported by the resource, a 415 (Unsupported Media Type) is appropriate. This specifically includes the case where the media type is known in principle, but lacks semantics specific to a QUERY to the target resource. In both cases, the Accept-Query response field (Section 3) can be used to inform the client of media types which are supported.

* If a media type is specified, but is inconsistent with the actual request content, a 400 (Bad Request) can be returned. That is, a server is not allowed to infer a media type from the request content and then override a missing or "erroneous" value ("content sniffing").

* If the media type is specified, is understood, and the content is indeed consistent with the type, but the query can not be processed due to the actual contents of the query, the status 422 (Unprocessable Content) can be used. An example would be a syntactically correct SQL query that identifies a non-existent table.

* Finally, if the client requests a specific response media type using the Accept field ([HTTP], Section 12.5.1) which is not supported by the resource, a status code of 406 (Not Acceptable) is appropriate.

2.2. Equivalent Resource

The _equivalent resource_ for any given QUERY request is a resource responding to GET requests, representing that QUERY request and its target, taking both message content and metadata into account (Section 6 of [HTTP]). In particular, this includes representation metadata (Section 8 of [HTTP]), such as the content’s media type.

In other words, the equivalent resource is derived from the resource implementing QUERY by incorporating the request content.

The term _equivalent resource_ is used as a means to define behavior for other HTTP aspects, such as selected representations. Servers can but do not have to assign URIs to these resources (see Section 1.1 of [URI]). If they do so, these resources will become accessible for GET requests.

2.3. Content-Location Response Field

A successful response (2xx, Section 15.3 of [HTTP]) can include a Content-Location header field containing an identifier for a resource corresponding to the results of the operation; see Section 8.7 of [HTTP] for details. This represents a claim from the server that a client can send a GET request for the indicated URI to retrieve the results of the query operation just performed. The indicated resource might be temporary.

See Appendix A.4.1 for an example.

2.4. Location Response Field

A server can assign a URI to the equivalent resource (Section 2.2) of a QUERY request. If the server does so, the URI of that resource can be included in the Location header field of the 2xx response (see Section 10.2.2 of [HTTP]). This represents a claim that a client can send a GET request to the indicated URI to repeat the query operation just performed without resending the query content. This resource's URI might be temporary; if a future request fails, the client can retry using the original QUERY request target and the previously submitted content.

See Appendix A.4.2 for an example.

2.5. Redirection

In some cases, the server may choose to respond indirectly to the QUERY request by redirecting the user agent to a different URI (see Section 15.4 of [HTTP]).

A response with either of the status codes 301 (Moved Permanently, [HTTP], Section 15.4.2) or 308 (Permanent Redirect, [HTTP], Section 15.4.9) indicates that the target resource has permanently moved to a different URI referenced by the Location response field ([HTTP], Section 10.2.2). Likewise, a response with either 302 (Found, [HTTP], Section 15.4.3 or 307 (Temporary Redirect, [HTTP], Section 15.4.8) indicates that the target resource has temporarily moved. In all four cases, the server is suggesting that the user agent can accomplish its original QUERY request by sending a similar QUERY request to the new target URI referenced by Location.

Note that the exceptions for redirecting a POST as a GET request after a 301 or 302 response do not apply to QUERY requests.

A response to QUERY with the status code 303 (See Other, Section 15.4.4 of [HTTP]) indicates that the original query can be accomplished via a normal retrieval request on the URI referenced by the Location response field ([HTTP], Section 10.2.2). For HTTP, this means sending a GET request to the new target URI, as illustrated by the example in Appendix A.4.3.

2.6. Conditional Requests

The selected representation (Section 3.2 of [HTTP]) of a QUERY request is the same as for a GET request to the equivalent resource (Section 2.2) of that QUERY request.

A conditional QUERY requests that that selected representation (i.e., the query results, after any content negotiation) be returned in the response only under the circumstances described by the conditional header field(s), as defined in Section 13 of [HTTP].

See Appendix A.5 for examples.

2.7. Caching

The response to a QUERY method is cacheable; a cache MAY use it to satisfy subsequent QUERY requests as per Section 4 of [HTTP-CACHING]).

The cache key for a QUERY request (see Section 2 of [HTTP-CACHING]) MUST incorporate the request content (Section 6 of [HTTP-CACHING]) and related metadata (Section 8 of [HTTP-CACHING]).

Caches MAY remove semantically insignificant differences first, thereby improving cache efficiency.

For instance, by

* removing content encoding(s) (Section 8.4 of [HTTP]).

* normalizing based upon knowledge of format conventions, as indicated by any media subtype suffix in the request’s Content-Type field (e.g., "+json", see Section 4.2.8 of [RFC6838]).

* normalizing based upon knowledge of the semantics of the content itself, as indicated by the request’s Content-Type field.

Note that any such transformation is performed solely for the purpose of generating a cache key; it does not change the request itself.

Clients can indicate, using the "no-transform" cache directive (Section 5.2.1.6 of [HTTP-CACHING]), that they wish that no such transformation happens (but note that this directive is just advisory).

Note that caching QUERY method responses is inherently more complex than caching responses to GET, as complete reading of the request's content is needed in order to determine the cache key. If a QUERY response supplies a Location response field (Section 2.4) to indicate a URI for an equivalent resource (Section 2.2), clients can switch to GET for subsequent requests, thereby simplifying processing.

2.8. Range Requests

The semantics of Range Requests for QUERY are identical to those for GET, as defined in Section 14 of [HTTP]. Byte Range requests (the only range unit defined at the time of writing), however, offer little value for the results of a QUERY request.

Query formats often define their own way for limiting or paging through result sets, such as with "FETCH FIRST ... ROWS ONLY" in SQL. It is expected that these built-in features will be used instead of HTTP Range Requests.

3. The "Accept-Query" Header Field

The "Accept-Query" response header field can be used by a resource to directly signal support for the QUERY method while identifying the specific query format media type(s) that may be used.

Accept-Query contains a list of media ranges (Section 12.5.1 of [HTTP]) using "Structured Fields" syntax ([STRUCTURED-FIELDS]). Media ranges are represented by a List Structured Header Field of either Tokens or Strings, containing the media range value without parameters.

Media type parameters, if any, are mapped to Structured Field Parameters of type String or Token. The choice of Token vs. String is semantically insignificant. That is, recipients MAY convert Tokens to Strings, but MUST NOT process them differently based on the received type.

Media types do not exactly map to Tokens, for instance they allow a leading digit. In cases like these, the String format needs to be used.

The only supported uses of wildcards are "*/*", which matches any type, or "xxxx/*", which matches any subtype of the indicated type.

The order of types listed in the field value is not significant.

The value of the Accept-Query field applies to every URI on the server that shares the same path; in other words, the query component is ignored. If requests to the same resource return different Accept-Query values, the most recently received fresh value (per Section 4.2 of [HTTP-CACHING]) is used.

Example:

Accept-Query: "application/jsonpath", application/sql;charset="UTF-8"

Although the syntax for this field appears to be similar to other fields, such as "Accept" (Section 12.5.1 of [HTTP]), it is a Structured Field and thus MUST be processed as specified in Section 4 of [STRUCTURED-FIELDS].

4. Security Considerations

The QUERY method is subject to the same general security considerations as all HTTP methods as described in [HTTP].

It can be used as an alternative to passing request information in the URI (e.g., in the query component). This is preferred in some cases, as the URI is more likely to be logged or otherwise processed by intermediaries than the request content. In other cases, where the query contains sensitive information, the potential for logging of the URI might motivate the use of QUERY over GET.

If a server creates a temporary resource to represent the results of a QUERY request (e.g., for use in the Location or Content-Location field) and the request contains sensitive information that cannot be logged, then the URI of this resource SHOULD be chosen such that it does not include any sensitive portions of the original request content.

Caches that normalize QUERY content incorrectly or in ways that are significantly different from how the resource processes the content can return an incorrect response if normalization results in a false positive.

A QUERY request from user agents implementing CORS (Cross-Origin Resource Sharing) will require a "preflight" request, as QUERY does not belong to the set of CORS-safelisted methods (see "Methods (https://fetch.spec.whatwg.org/#methods)" in [FETCH]).

5. IANA Considerations

5.1. Registration of QUERY method

IANA is requested to add the QUERY method to the HTTP Method Registry at <http://www.iana.org/assignments/http-methods> (see Section 16.3.1 of [HTTP]).

Method Name	Safe	Idempotent	Specification
QUERY	Yes	Yes	Section 2

Table 2: QUERY Method Definition

5.2. Registration of Accept-Query field

IANA is requested to add the Accept-Query field to the HTTP Field Name Registry at <https://www.iana.org/assignments/http-fields> (see Section 16.1.1 of [HTTP]).

Field Name	Status	Structured Type	Reference	Comments
Accept-Query	permanent	List	Section 3 of this document.	

Table 3: Accept-Query Field Definition

6. Normative References

[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>.

[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>.

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, 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, June 2022, <https://www.rfc-editor.org/rfc/rfc9111>.

[STRUCTURED-FIELDS]
Nottingham, M. and P-H. Kamp, "Structured Field Values for HTTP", RFC 9651, September 2024, <https://www.rfc-editor.org/rfc/rfc9651>.

7. Informative References

[FETCH] WHATWG, "FETCH", <https://fetch.spec.whatwg.org>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC9535] G旦ssner, S., Ed., Normington, G., Ed., and C. Bormann, Ed., "JSONPath: Query Expressions for JSON", RFC 9535, DOI 10.17487/RFC9535, February 2024, <https://www.rfc-editor.org/info/rfc9535>.

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

[URL] WHATWG, "URL", <https://url.spec.whatwg.org>.

[XSLT] Kay, M., "XSL Transformations (XSLT) Version 3.0", W3C Recommendation REC-xslt-30-20170608, 8 June 2017, <https://www.w3.org/TR/2017/REC-xslt-30-20170608/>. Latest version available at https://www.w3.org/TR/xslt-30/.

Appendix A. Examples

The examples below are for illustrative purposes only; if one needs to send queries that are actually this short, it is likely better to use GET.

The media type used in most examples is "application/x-www-form-urlencoded" (as used in POST requests from browser user clients, defined in "application/x-www-form-urlencoded (https://url.spec.whatwg.org/#application/x-www-form-urlencoded)" in [URL]). The Content-Length fields have been omitted for brevity.

A.1. Simple Query

A simple query with a direct response:

```
QUERY /contacts HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded
Accept: application/json

select=surname,givenname,email&limit=10&match=%22email=*@example.*%22
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  { "surname": "Smith",
    "givenname": "John",
    "email": "smith@example.org" },
  { "surname": "Jones",
    "givenname": "Sally",
    "email": "sally.jones@example.com" },
  { "surname": "Dubois",
    "givenname": "Camille",
    "email": "camille.dubois@example.net" }
]
```

A.2. Discovery of QUERY support

A simple way to discover support for QUERY is provided by the OPTIONS (Section 9.3.7 of [HTTP]) method:

```
OPTIONS /contacts HTTP/1.1
Host: example.org
```

Response:

```
HTTP/1.1 200 OK
Allow: GET, QUERY, OPTIONS, HEAD
```

The Allow response field (Section 10.2.1 of [HTTP]) denotes the set of supported methods on the specified resource.

There are alternatives to the use of OPTIONS. For instance, a QUERY request can be tried without prior knowledge of server support. The server would then either process the request, or could respond with a 4xx status such as 405 (Method Not Allowed, Section 15.5.6 of [HTTP]), including the Allow response field.

A.3. Discovery of QUERY Formats

Discovery of supported media types for QUERY is possible via the Accept-Query (Section 3) response field:

```
HEAD /contacts HTTP/1.1
Host: example.org
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/xhtml
Accept-Query: application/x-www-form-urlencoded, application/sql
```

Responses to which request methods will contain Accept-Query will depend on the resource being accessed.

An alternative to checking Accept-Query would be to make a QUERY request, and then -- in case of a 4xx status such as 415 (Unsupported Media Type, Section 15.5.16 of [HTTP]) response -- to inspect the Accept (Section 12.5.1 of [HTTP]) response field:

```
HTTP/1.1 415 Unsupported Media Type
Content-Type: application/xhtml
Accept: application/x-www-form-urlencoded, application/sql
```

A.4. Content-Location, Location, and Indirect Responses

As described in Sections 2.3 and 2.4, the Content-Location and Location response fields in success responses (2xx, Section 15.3 of [HTTP]) provide a way to identify alternate resources that will respond to GET requests, either for the received result of the request, or for future requests to perform the same operation. Going back to the example from Appendix A.1:

```
QUERY /contacts HTTP/1.1
Host: example.org
Content-Type: application/x-www-form-urlencoded
Accept: application/json

select=surname,givenname,email&limit=10&match=%22email=*@example.*%22
```

Response:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Location: /contacts/stored-results/17
Location: /contacts/stored-queries/42
Last-Modified: Sat, 25 Aug 2012 23:34:45 GMT
Date: Sun, 17 Nov 2024, 16:10:24 GMT

[
  { "surname": "Smith",
    "givenname": "John",
    "email": "smith@example.org" },
  { "surname": "Jones",
    "givenname": "Sally",
    "email": "sally.jones@example.com" },
  { "surname": "Dubois",
    "givenname": "Camille",
    "email": "camille.dubois@example.net" }
]
```

A.4.1. Using Content-Location

The Content-Location response field received above identifies a resource holding the result for the QUERY response it appeared on:

```
GET /contacts/stored-results/17 HTTP/1.1
Host: example.org
Accept: application/json
```

Response:

```
HTTP/1.1 200 OK
Last-Modified: Sat, 25 Aug 2012 23:34:45 GMT
Date: Sun, 17 Nov 2024, 16:10:25 GMT

[
  { "surname": "Smith",
    "givenname": "John",
    "email": "smith@example.org" },
  { "surname": "Jones",
    "givenname": "Sally",
    "email": "sally.jones@example.com" },
  { "surname": "Dubois",
    "givenname": "Camille",
    "email": "camille.dubois@example.net" }
]
```

Note that there’s no guarantee that the server will implement this resource indefinitely, so, after an error response, the client would need to redo the original QUERY request in order to obtain a new alternative location.

A.4.2. Using Location

The Location response field identifies a resource that will respond to GET with a current result for the same process and parameters as the original QUERY request.

```
GET /contacts/stored-queries/42 HTTP/1.1
Host: example.org
Accept: application/json
```

In this example, one entry was removed at 2024-11-17T16:12:01Z (as indicated in the Last-Modified field), so the response only contains two entries:

```
HTTP/1.1 200 OK
Content-Type: application/json
Last-Modified: Sun, 17 November 2024, 16:12:01 GMT
ETag: "42-1"
Date: Sun, 17 Nov 2024, 16:13:17 GMT

[
  { "surname": "Smith",
    "givenname": "John",
    "email": "smith@example.org" },
  { "surname": "Dubois",
    "givenname": "Camille",
    "email": "camille.dubois@example.net" }
]
```

Assuming that the server still exposes the resource and that there was no change in the query result, a subsequent conditional GET request with

```
If-None-Match: "42-1"
```

would result in a 304 response (Not Modified, Section 15.4.5 of [HTTP]).

A.4.3. Indirect Responses

Servers can send "indirect" responses (Section 2.5) using the status code 303 (See Other, Section 15.4.4 of [HTTP]).

Given the request at the beginning of Appendix A.4, a server might respond with:

```
HTTP/1.1 303 See Other
Content-Type: text/plain
Date: Sun, 17 Nov 2024, 16:13:17 GMT
Location: /contacts/stored-queries/42

See stored query at "/contacts/stored-queries/42".
```

This is similar to including Location on a direct response, except that no result for the query is returned. This allows the server to only generate or reuse an alternative resource. This resource could then be used as shown in Appendix A.4.2.

A.5. Conditional Requests

Consider a resource implementing QUERY that supports "application/sql" and "application/xslt+xml" ([XSLT]) as request media types, and which can generate responses as "text/csv" . The data set being queried contains RFC document information, and the query returns information grouped by decade:

```
QUERY /rfc-index.xml HTTP/1.1
Host: example.org
Date: Sun, 7 Sep 2025, 00:00:00 GMT
Content-Type: application/xslt+xml
Accept: text/csv

...Query content using XSLT...
```

Response:

```
HTTP/1.1 200 OK
Date: Sun, 7 Sep 2025, 00:00:00 GMT
Location: /stored-queries/4815162342
Content-Type: text/csv
Accept-Query: "application/sql", "application/xslt+xml"
Last-Modified: Sun, 31 Aug 2025, 08:44:00 GMT
Vary: Accept-Query, Content-Encoding, Content-Type

decade, total, with errata, % with errata, average page count
1960, 26, 5, 19.2, 5.3
1970, 666, 18, 2.7, 6.1
1980, 376, 44, 11.7, 23.4
1990, 1593, 269, 16.9, 25.5
2000, 2888, 1048, 36.3, 27.3
2010, 2954, 895, 30.3, 26.1
2020, 1133, 230, 20.3, 26.2
```

Here, the server has assigned the path "/stored-queries/4815162342" to the equivalent resource (Section 2.4) for subsequent use with GET.

Later on, the client repeats the query, but specifies that results should only be returned when changed:

```
QUERY /rfc-index.xml HTTP/1.1
Host: example.org
Date: Mon, 8, Sep 2025, 11:00:00 GMT
Content-Type: application/sql
Accept: text/csv
If-Modified-Since: Sun, 31 Aug 2025, 08:44:00 GMT
Vary: Accept-Query, Content-Type

...Same query, but using SQL...
```

The data being queried did not change, therefore the server responds with:

```
HTTP/1.1 304 Not Modified
Date: Mon, 8 Sep 2025, 11:00:00 GMT
Content-Type: text/csv
Location: /stored-queries/4815162342
Accept-Query: "application/sql", "application/xslt+xml"
Last-Modified: Sun, 31 Aug 2025, 08:44:00 GMT
Vary: Accept-Query, Content-Type
```

As the server identified a URI for the equivalent resource, that resource can be accessed with GET. In particular, this avoids re-sending the query request's content:

```
GET /stored-queries/4815162342 HTTP/1.1
Host: example.org
Date: Sun, 21, Sep 2025, 12:08:00 GMT
Accept: text/csv
If-Modified-Since: Sun, 31 Aug 2025, 00:00:00 GMT
```

Here, the state of the data set indeed changed, so new content is returned:

```
HTTP/1.1 200 OK
Date: Sun, 21, Sep 2025, 12:08:00 GMT
Content-Type: text/csv
Last-Modified: Thu, 18 Sep 2025, 19:56:00 GMT
Vary: Accept-Query, Content-Encoding, Content-Type

decade, total, with errata, % with errata, average page count
1960, 26, 5, 19.2, 5.3
1970, 666, 18, 2.7, 6.1
1980, 376, 44, 11.7, 23.4
1990, 1593, 269, 16.9, 25.5
2000, 2888, 1048, 36.3, 27.3
2010, 2954, 895, 30.3, 26.1
2020, 1133, 230, 20.3, 26.2
```

(Note the change in the row for this decade.)

The diagrams below illustrate the use of conditional requests and how they can differ when a URI is assigned to the equivalent resource (and when the client is taking advantage of it). The fictitious field name "Validator" is used for demonstration purposes.

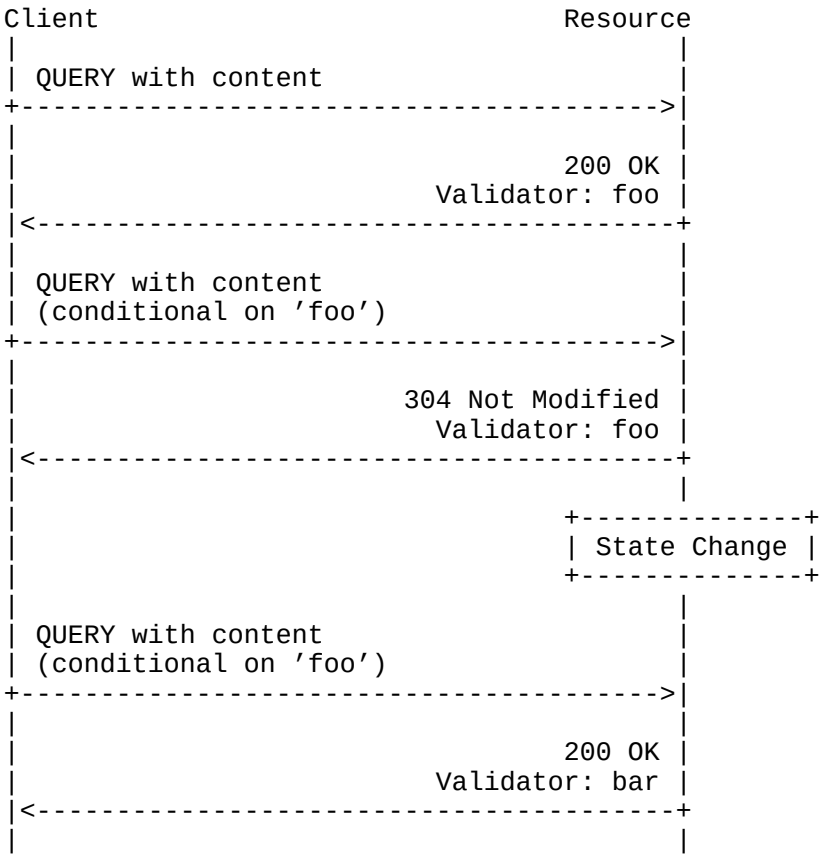

Figure 1: Data Flow with QUERY only

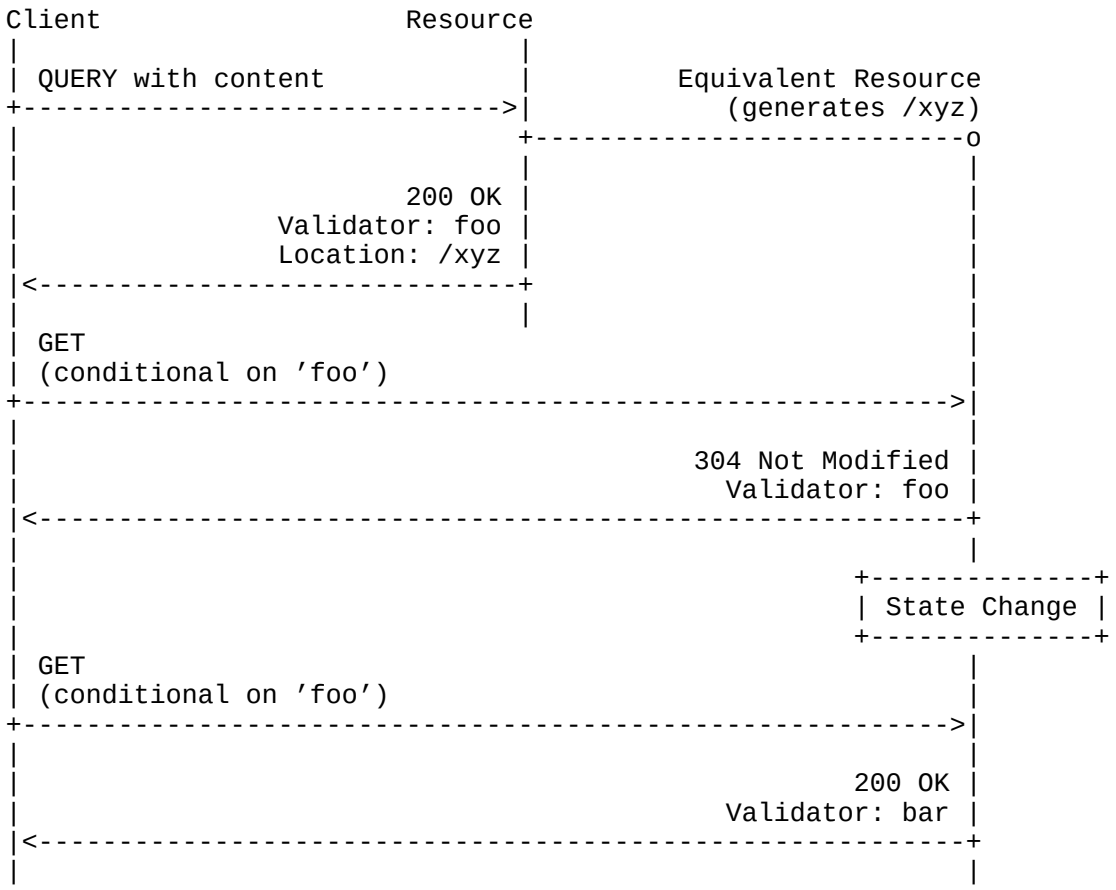

Figure 2: Data Flow with GET to Equivalent Resource

A.6. More Query Formats

The following examples show requests on a JSON-shaped ([RFC8259]) database of RFC errata.

The request below uses XSLT to extract errata information summarized per year and the defined errata types.

```
QUERY /errata.json HTTP/1.1
Host: example.org
Content-Type: application/xslt+xml
Accept: application/xml, text/csv

<transform xmlns="http://www.w3.org/1999/XSL/Transform"
  xmlns:j="http://www.w3.org/2005/xpath-functions"
  version="3.0">

  <output method="text"/>

  <param name="input"/>

  <variable name="json"
    select="json-to-xml(unparsed-text($input))"/>

  <variable name="sc">errata_status_code</variable>
  <variable name="sd">submit_date</variable>

  <template match="/">
    <text>year, total, rejected, verified, hdu, reported</text>
    <text>&#10;</text>
    <variable name="en" select="$json//j:map"/>
    <for-each-group select="$en"
      group-by="substring-before(j:string[@key=$sd],'-')">
      <sort select="current-grouping-key()"/>
      <variable name="year" select="current-grouping-key()"/>
      <variable name="errata" select=
        "$en[$year=substring-before(j:string[@key=$sd],'-')]"/>
      <value-of select="concat(
        $year,
        ',',
        count($errata),
        ',',
        count($errata['Rejected'=j:string[@key=$sc]]),
        ',',
        count($errata['Verified'=j:string[@key=$sc]]),
        ',',
        count(
          $errata['Held for Document Update'=j:string[@key=$sc]]),
        ',',
        count($errata['Reported'=j:string[@key=$sc]]),
        '&#10;')"/>
    </for-each-group>
  </template>

</transform>
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/csv
Accept-Query: "application/jsonpath", "application/xslt+xml"
Date: Wed, 19 Feb 2025, 17:10:01 GMT

year, total, rejected, verified, hdu, reported
2000, 14, 0, 14, 0, 0
2001, 72, 1, 70, 1, 0
2002, 124, 8, 104, 12, 0
2003, 63, 0, 61, 2, 0
2004, 89, 1, 83, 5, 0
2005, 156, 10, 96, 50, 0
2006, 444, 54, 176, 214, 0
2007, 429, 48, 188, 193, 0
2008, 423, 52, 165, 206, 0
2009, 331, 39, 148, 144, 0
2010, 538, 80, 232, 222, 4
2011, 367, 47, 170, 150, 0
2012, 348, 54, 149, 145, 0
2013, 341, 61, 169, 106, 5
2014, 342, 73, 180, 72, 17
2015, 343, 79, 145, 89, 30
2016, 295, 46, 122, 82, 45
2017, 303, 46, 120, 84, 53
2018, 350, 61, 118, 98, 73
2019, 335, 47, 131, 94, 63
2020, 387, 68, 117, 123, 79
2021, 321, 44, 148, 63, 66
2022, 358, 37, 198, 40, 83
2023, 262, 38, 121, 33, 70
2024, 322, 33, 125, 23, 141
9999, 1, 0, 0, 1, 0
```

Note the Accept-Query response field indicating that another query format -- JSONPath ([RFC9535]) -- is supported as well. The request below would report the identifiers of all rejected errata submitted since 2024:

```
QUERY /errata.json HTTP/1.1
Host: example.org
Content-Type: application/jsonpath
Accept: application/json

$..[
     ?@.errata_status_code=="Rejected"
     && @.submit_date>"2024"
   ]
   ["doc-id"]

Response:

HTTP/1.1 200 OK
Content-Type: application/json
Accept-Query: "application/jsonpath", "application/xslt+xml"
Date: Thu, 20 Feb 2025, 09:55:42 GMT
Last-Modified: Thu, 20 Feb 2025 06:10:01 GMT

[
  "RFC1185","RFC8407","RFC6350","RFC8467","RFC1157","RFC9543",
  "RFC9076","RFC7656","RFC2822","RFC9460","RFC2104","RFC6797",
  "RFC9499","RFC9557","RFC2131","RFC2328","RFC9001","RFC3325",
  "RFC9438","RFC2526","RFC2985","RFC7643","RFC9132","RFC6376",
  "RFC9110","RFC9460","RFC7748","RFC9497","RFC8463","RFC4035",
  "RFC7239","RFC9083","RFC9537","RFC9537","RFC9420","RFC9000",
  "RFC9656","RFC9110","RFC2324","RFC2549","RFC6797","RFC2549",
  "RFC8894"
]
```

Appendix B. Change Log

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

B.1. Since draft-ietf-httpbis-safe-method-w-body-00

* Use "example/query" media type instead of undefined "text/query" (https://github.com/httpwg/http-extensions/issues/1450)

* In Section 3, adjust the grammar to just define the field value (https://github.com/httpwg/http-extensions/issues/1470)

* Update to latest HTTP core spec, and adjust terminology accordingly (https://github.com/httpwg/http-extensions/issues/1473)

* Reference RFC 8174 and markup bcp14 terms (https://github.com/httpwg/http-extensions/issues/1497)

* Update HTTP reference (https://github.com/httpwg/http-extensions/issues/1524)

* Relax restriction of generic XML media type in request content (https://github.com/httpwg/http-extensions/issues/1535)

B.2. Since draft-ietf-httpbis-safe-method-w-body-01

* Add minimal description of cacheability (https://github.com/httpwg/http-extensions/issues/1552)

* Use "QUERY" as method name (https://github.com/httpwg/http-extensions/issues/1614)

* Update HTTP reference (https://github.com/httpwg/http-extensions/issues/1669)

B.3. Since draft-ietf-httpbis-safe-method-w-body-02

* In Section 3, slightly rephrase statement about significance of ordering (https://github.com/httpwg/http-extensions/issues/1896)

* Throughout: use "content" instead of "payload" or "body" (https://github.com/httpwg/http-extensions/issues/1915)

* Updated references (https://github.com/httpwg/http-extensions/issues/2157)

B.4. Since draft-ietf-httpbis-safe-method-w-body-03

* In Section 3, clarify scope (https://github.com/httpwg/http-extensions/issues/1913)

B.5. Since draft-ietf-httpbis-safe-method-w-body-04

* Describe role of Content-Location and Location fields (https://github.com/httpwg/http-extensions/issues/1745)

* Added Mike Bishop as author (https://github.com/httpwg/http-extensions/issues/2837)

* Use "target URI" instead of "effective request URI" (https://github.com/httpwg/http-extensions/issues/2883)

B.6. Since draft-ietf-httpbis-safe-method-w-body-05

* Updated language and examples about redirects and method rewriting (https://github.com/httpwg/http-extensions/issues/1917)

* Add QUERY example to introduction (https://github.com/httpwg/http-extensions/issues/2171)

* Update "Sensitive information in QUERY URLs" (https://github.com/httpwg/http-extensions/issues/2853)

* Field registration for "Accept-Query" (https://github.com/httpwg/http-extensions/issues/2903)

B.7. Since draft-ietf-httpbis-safe-method-w-body-06

* Improve language about sensitive information in URIs (https://github.com/httpwg/http-extensions/issues/1895)

* Guidance about what’s possible with GET wrt URI length (https://github.com/httpwg/http-extensions/issues/1914)

* Clarified description of conditional queries (https://github.com/httpwg/http-extensions/issues/1917)

* Editorial changes to Introduction (ack Will Hawkins, https://github.com/httpwg/http-extensions/pull/2859)

* Added Security Consideration with respect to Normalization (https://github.com/httpwg/http-extensions/issues/2896)

* Added CORS considerations (https://github.com/httpwg/http-extensions/issues/2898)

* Make Accept-Query a Structured Field (https://github.com/httpwg/http-extensions/issues/2934)

* SQL media type is application/sql (RFC6922) (https://github.com/httpwg/http-extensions/issues/2936)

* Added overview table to introduction (https://github.com/httpwg/http-extensions/issues/2951)

* Reference HTTP spec for terminology (https://github.com/httpwg/http-extensions/issues/2953)

* Moved BCP14 related text into subsection (https://github.com/httpwg/http-extensions/issues/2954)

* Move examples into index (https://github.com/httpwg/http-extensions/issues/2957)

B.8. Since draft-ietf-httpbis-safe-method-w-body-07

* Examples Section revised (https://github.com/httpwg/http-extensions/issues/1906)

* Discuss Range Requests (https://github.com/httpwg/http-extensions/issues/2979)

B.9. Since draft-ietf-httpbis-safe-method-w-body-08

* Mention the role of the query part of the request URI (https://github.com/httpwg/http-extensions/issues/3004)

* Avoid term ’query parameters’ (https://github.com/httpwg/http-extensions/issues/3019)

* Add missing references, fixed terminology (https://github.com/httpwg/http-extensions/issues/3021)

* Add Acknowledgements/Contributors sections; moved Ashok to Contributors (https://github.com/httpwg/http-extensions/issues/3029)

* Hopefully more clarity wrt query content vs URI query component (https://github.com/httpwg/http-extensions/issues/3059)

B.10. Since draft-ietf-httpbis-safe-method-w-body-09

* Clarify cacheability of POST (https://github.com/httpwg/http-extensions/issues/3068)

* Rephrase text that suggests a media type definition can override URI semantics (https://github.com/httpwg/http-extensions/issues/3069)

* Restrict description of Content-Location and Location semantics to 2xx responses (https://github.com/httpwg/http-extensions/issues/3070)

* Slightly rephrase semantics for Content-Location (https://github.com/httpwg/http-extensions/issues/3071)

B.11. Since draft-ietf-httpbis-safe-method-w-body-10

* Editorial nits (https://github.com/httpwg/http-extensions/pull/3080, ack martinthomson)

* Fix references in Appendix A.3 (https://github.com/httpwg/http-extensions/pull/3090, ack Rahul Gupta)

* Update James' affiliation (https://github.com/httpwg/http-extensions/pull/3094)

* Review references to HTTP (https://github.com/httpwg/http-extensions/pull/3097)

* Address most Rahul Gupta's additional feedback (https://github.com/httpwg/http-extensions/pull/3101)

B.12. Since draft-ietf-httpbis-safe-method-w-body-11

* Improve description of caching, clarifying what is required (https://github.com/httpwg/http-extensions/tree/reschke-3107)

* Address HTTPDIR/RF feedback on example appendix (https://github.com/httpwg/http-extensions/issues/3114)

* Address HTTPDIR/RF feedback on redirection (https://github.com/httpwg/http-extensions/issues/3119)

* Address HTTPDIR/RF feedback on caching (https://github.com/httpwg/http-extensions/issues/3120)

* Address HTTPDIR/RF feedback on abstract (https://github.com/httpwg/http-extensions/issues/3121)

* Address HTTPDIR/RF feedback on introduction (https://github.com/httpwg/http-extensions/issues/3122)

* Address HTTPDIR/RF feedback on method definition (https://github.com/httpwg/http-extensions/issues/3123)

* Consistent Table Captions (https://github.com/httpwg/http-extensions/issues/3134)

* Define "Equivalent Resource", update description of Conditional Requests, add examples (https://github.com/httpwg/http-extensions/issues/3137)

* Extend discussion of Range Requests (https://github.com/httpwg/http-extensions/issues/3151)

Acknowledgements

We thank all members of the HTTP Working Group for ideas, reviews, and feedback.

The following individuals deserve special recognition: Carsten Bormann, Mark Nottingham, Martin Thomson, Michael Thornburgh, Roberto Polli, Roy Fielding, and Will Hawkins.

Contributors

Ashok Malhotra participated in early discussions leading to this specification:

Ashok Malhotra
Email: malhotrasahib@gmail.com

Authors' Addresses

Julian Reschke
greenbytes GmbH
Hafenweg 16
48155 M端nster
Germany
Email: julian.reschke@greenbytes.de
URI: https://greenbytes.de/tech/webdav/

James M Snell
Cloudflare
Email: jasnell@gmail.com

Mike Bishop
Akamai
Email: mbishop@evequefou.be