Workgroup: HTTP
Published: 4 July 2022
Intended Status: Standards Track
Expires: 5 January 2023
Authors: J. Reschke   A. Malhotra   J.M. Snell
greenbytes

The HTTP QUERY Method

Abstract

This specification defines a new HTTP method, QUERY, as a safe, idempotent request method that can carry request content.

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/safe-method-w-body.

The changes in this draft are summarized in Appendix A.3.

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 that contains content.

Most often, this is desirable when the data conveyed in a request is too voluminous to be encoded into the request's URI. For example, while this is an common and interoperable query:

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

if the query parameters extend to several kilobytes or more of data it may not be, because many implementations place limits on their size. Often these limits are not known or discoverable ahead of time, because a request can pass through many uncoordinated systems. Additionally, expressing some data in the target URI is inefficient, because it needs to be encoded to be a valid URI.

Encoding query parameters directly into the request URI also effectively casts every possible combination of query inputs as distinct resources. Depending on the application, that may not be desirable.

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 parameters to the search operation are passed along within the request content as opposed to using the request URI.

A typical use of HTTP POST for requesting a search

```
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 same basic limitation as GET in 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. As with POST, the input to the query operation is passed along within 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.

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 HTTP GET method, which requests that a server return 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 server to perform a query operation (described by the request content) over some set of data scoped to the effective request URI. The content returned in response to a QUERY cannot be assumed to be a representation of the resource identified by the effective request URI.

The content of the request defines the query. Implementations **MAY** use a request content of any media type with the QUERY method, provided that it has appropriate query semantics.

QUERY requests are both safe and idempotent with regards to the resource identified by the request URI. That is, QUERY requests do not alter the state of the targeted resource. However, while processing a QUERY request, a server can be expected to allocate computing and memory resources or even create additional HTTP resources through which the response can be retrieved.

A successful response to a QUERY request is expected to provide some indication as to the final disposition of the operation. For instance, a successful query that yields no results can be represented by a 204 No Content response. If the response includes content, it is expected to describe the results of the operation. In some cases, the server may choose to respond indirectly to the QUERY request by returning a 3xx Redirection with a Location header field specifying an alternate Request URI from which the results can be retrieved using an HTTP GET request. Various non-normative examples of successful QUERY responses are illustrated in Section 4.

The semantics of the QUERY method change to a "conditional QUERY" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field ([HTTP], Section 13). A conditional QUERY requests that the query be performed only under the circumstances described by the conditional header field(s). It is important to note, however, that such conditions are evaluated against the state of the target resource itself as opposed to the collected results of the search operation.

2.1. 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 (see Section 2 of [HTTP-CACHING]) **MUST** incorporate the request content. When doing so, caches **SHOULD** first normalize request content to remove semantically insignificant differences, thereby improving cache efficiency, by:

* Removing content encoding(s)

* Normalizing based upon knowledge of format conventions, as indicated by the any media type suffix in the request's Content-Type field (e.g., "+json")

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

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

3. The "Accept-Query" Header Field

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

```
Accept-Query = 1#media-type
```

The Accept-Query header field specifies a comma-separated listing of media types (with optional parameters) as defined by Section 8.3.1 of [HTTP].

The order of types listed by the Accept-Query header field is not significant.

4. Examples

The non-normative examples in this section make use of a simple, hypothetical plain-text based query syntax based on SQL with results returned as comma-separated values. This is done for illustration purposes only. Implementations are free to use any format they wish on both the request and response.

4.1. Simple QUERY with a Direct Response

A simple query with a direct response:

```
QUERY /contacts HTTP/1.1
Host: example.org
Content-Type: example/query
Accept: text/csv

select surname, givenname, email limit 10
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/csv

surname, givenname, email
Smith, John, john.smith@example.org
Jones, Sally, sally.jones@example.com
Dubois, Camille, camille.dubois@example.net
```

4.2. Simple QUERY with indirect response (303 See Other)

A simple query with an Indirect Response (303 See Other):

```
QUERY /contacts HTTP/1.1
Host: example.org
Content-Type: example/query
Accept: text/csv

select surname, givenname, email limit 10
```

Response:

```
HTTP/1.1 303 See Other
Location: http://example.org/contacts/query123
```

Fetch Query Response:

```
GET /contacts/query123 HTTP/1.1
Host: example.org
```

Response:

```
HTTP/1.1 200 OK
Content-Type: text/csv

surname, givenname, email
Smith, John, john.smith@example.org
Jones, Sally, sally.jones@example.com
Dubois, Camille, camille.dubois@example.net
```

5. Security Considerations

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

6. IANA Considerations

IANA is requested to add QUERY method in the permanent registry at <http://www.iana.org/assignments/http-methods> (see Section 16.1.1 of [HTTP]).

Method Name	Safe	Idempotent	Specification
QUERY	Yes	Yes	Section 2

Table 1

7. 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>.

Appendix A. Change Log

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

A.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)

A.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)

A.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)

Authors' Addresses

Julian Reschke
greenbytes GmbH
Hafenweg 16
48155 Münster
Germany

Email: julian.reschke@greenbytes.de
URI: https://greenbytes.de/tech/webdav/

Ashok Malhotra

Email: malhotrasahib@gmail.com

James M Snell

Email: jasnell@gmail.com