CoRE
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

C. Bormann
Universitaet Bremen TZI

October 19, 2015

CoAP FETCH Method
draft-bormann-core-coap-fetch-00

Abstract

Similar to HTTP, the existing Constrained Application Protocol (CoAP) GET method only allows the specification of a URI and request parameters in CoAP options, not the transfer of a request payload detailing the request. This leads to some applications to using POST where actually a cacheable, idempotent, safe request is desired.

The present proposal adds a new CoAP method, FETCH, to perform the equivalent of a GET with a request body.

This specification is inspired by I-D.snell-search-method, which updates the definition and semantics of the HTTP SEARCH request method previously defined by RFC5323. However, there is no intention to limit FETCH to search-type operations, and the resulting properties may not be the same as those of HTTP SEARCH. For now, we therefore prefer to discuss the proposal under a different name, for which we have chosen the GET synonym FETCH.

Status of This Memo

Table of Contents

1. Introduction

The CoAP GET method [RFC7252] is used to obtain the representation of a resource, where the resource is specified by a URI and additional request parameters can additionally shape the representation. This has been modelled after the HTTP GET operation and the REST model in general.

In HTTP, a resource is often used to search for information, and existing systems varyingly use the HTTP GET and POST methods to perform a search. Often a POST method is used solely to enable supplying a larger set of parameters to the search than can be comfortably transferred in the URI with a GET request. [I-D.snell-search-method] proposes a SEARCH method that is similar to GET in most properties but enables sending a request body as with POST.

A major problem with GET is that the information that controls the request needs to be bundled up in some unspecified way into the URI.

Using the request body for this information has a number of advantages:

- The client can specify a media type (and a content encoding), enabling the server to unambiguously interpret the request parameters in the context of that media type. Also, the request body is not limited by the character set limitations of URIs, enabling a more natural (and more efficient) representation of certain domain-specific parameters.

- The request parameters are not limited by the maximum size of the URI. In HTTP, that is a problem as the practical limit for this size varies. In CoAP, another problem is that the block-wise transfer is not available for transferring large URI options in multiple rounds.

As an alternative to using GET, many implementations make use of the POST method to perform extended requests, even if they are semantically idempotent, safe, and even cacheable, to be able to pass along the input parameters within the request payload as opposed to using the request URI.

The FETCH method provides a solution that spans the gap between the use of GET and POST. As with POST, the input to the FETCH operation is passed along within the payload of the request rather than as part of the request URI. Unlike POST, however the semantics of the FETCH method are more specifically defined.

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 [RFC2119].

2. FETCH

The CoAP FETCH method is used to obtain a representation of a resource, giving a number of request parameters. Unlike the CoAP GET method, which requests that a server return a representation of the resource identified by the effective request URI (as defined by [RFC7252]), the FETCH method is used by a client to ask the server to produce a representation based on the request parameters (described by the request options and payload) based on the resource specified by the effective request URI. The payload returned in response to a FETCH cannot be assumed to be a complete representation of the resource identified by the effective request URI.

The body of the request defines the request parameters. Implementations MAY use a request body of any content type with the

FETCH method; it is outside the scope of this document how information about admissible content types is obtained by the client (although we can hint that form relations ([I-D.hartke-core-apps]) might be the preferred way).

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

A successful response to a FETCH request is expected to provide some indication as to the final disposition of the requested operation. If the response includes a body payload, the payload is expected to describe the results of the FETCH operation.

Depending on the response code as defined by [RFC7252] the response to a FETCH request is cacheable; the request payload is part of the cache key. Specifically, 2.05 "Content" response codes, the responses for which are cacheable, are a usual way to respond to a FETCH request. (Note that this aspect differs markedly from [I-D.snell-search-method].) (Note also that caches that cannot use the request payload as part of the request key will not be able to cache responses to FETCH requests at all.) The Max-Age option in the response has equivalent semantics to its use in a GET.

The semantics of the FETCH method change to a "conditional FETCH" if the request message includes an If-Match, or If-None-Match option ([RFC7252]). A conditional FETCH requests that the query be performed only under the circumstances described by the conditional option(s). It is important to note, however, that such conditions are evaluated against the state of the target resource itself as opposed to the results of the FETCH operation. [[This needs some additional text on what an ETag on a FETCH result means.]]

2.1. The Content-Format Option

A FETCH request MUST include a Content-Format option to specify the media type and content encoding of the request body.

2.2. Working with Observe

The Observe option [RFC7641] can be used with a FETCH request as it can be used with a GET request.

2.3. Working with Block

The Block1 option [I-D.ietf-core-block] can be used with a FETCH request as it would be used with a POST request; the Block2 option can then be used as with GET or POST.

2.4. Discussion

One property of FETCH that may be non-obvious is that a FETCH request cannot be generated from a link alone, but also needs a way to generate the request payload. Again, form relations ([I-D.hartke-core-apps]) may be able to fill parts of this gap.

3. Security Considerations

The FETCH method is subject to the same general security considerations as all CoAP methods as described in [RFC7252].

4. IANA Considerations

IANA is requested to add an entry to the sub-registry "CoAP Method Codes":

Code	Name	Reference
0.07	FETCH	[RFCthis]

The FETCH method is idempotent and safe, and it returns the same response codes that GET can return, plus 4.15 "Unsupported Content-Format" with the same semantics as with POST.

5. Acknowledgements

Most of the text in this I-D was stolen, e.g. from [I-D.snell-search-method] or from [I-D.vanderstok-core-patch]. Thank you!

Klaus Hartke found a number of problems while quickly checking an earlier version of this document.

6. References

6.1. Normative References

[I-D.ietf-core-block]
Bormann, C. and Z. Shelby, "Block-wise transfers in CoAP", draft-ietf-core-block-18 (work in progress), September 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <http://www.rfc-editor.org/info/rfc7641>.

6.2. Informative References

[I-D.hartke-core-apps] Hartke, K., "CoRE Application Descriptions", draft-hartke-core-apps-02 (work in progress), August 2015.

[I-D.snell-search-method] Reschke, J., Malhotra, A., and J. Snell, "HTTP SEARCH Method", draft-snell-search-method-00 (work in progress), April 2015.

[I-D.vanderstok-core-patch] Stok, P. and A. Sehgal, "Patch Method for Constrained Application Protocol (CoAP)", draft-vanderstok-core-patch-02 (work in progress), October 2015.

Author’s Address

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63921
Email: cabo@tzi.org