

core
Internet-Draft
Intended status: Informational
Expires: September 22, 2016

P. van der Stok
Consultant
C. Bormann
Universitaet Bremen TZI
A. Sehgal
Consultant
March 21, 2016

Patch and Fetch Methods for Constrained Application Protocol (CoAP)
draft-vanderstok-core-etch-00

Abstract

The existing Constrained Application Protocol (CoAP) methods only allow access to a complete resource. This does not permit applications to access parts of a resource. In case of resources with larger or complex data, or in situations where a resource continuity is required, replacing or requesting the whole resource is undesirable. Several applications using CoAP will need to perform partial resource accesses.

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.

Again similar to HTTP, the existing Constrained Application Protocol (CoAP) PUT method only allows to replace a complete resource. This also leads applications to use POST where actually a cacheable, possibly idempotent request is desired.

This specification adds new CoAP methods, FETCH, to perform the equivalent of a GET with a request body; and the twin methods PATCH and iPATCH, to modify parts of an existing CoAP resource.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 22, 2016.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	FETCH	3
1.2.	PATCH and iPATCH	4
1.3.	Requirements Language	4
1.4.	Terminology and Acronyms	4
2.	FETCH Method	4
2.1.	The Content-Format Option	6
2.2.	Working with Observe	6
2.3.	Working with Block	6
2.4.	FETCH discussion	6
3.	PATCH and iPATCH Methods	6
3.1.	Simple Examples for PATCH and iPATCH	8
3.2.	Response Codes	10
3.3.	Option Numbers	10
3.4.	Error Handling	10
4.	Discussion	12
5.	Security Considerations	12
6.	IANA Considerations	13
7.	Acknowledgements	14
8.	Change log	14
9.	References	14
9.1.	Normative References	14
9.2.	Informative References	15
	Authors' Addresses	16

1. Introduction

This specification defines the new Constrained Application Protocol (CoAP) [RFC7252] methods, FETCH, PATCH and iPATCH, which are used to access and update parts of a resource.

1.1. FETCH

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 for the sole reason that a larger set of parameters to the search can be supplied in the request body than can comfortably be transferred in the URI with a GET request. The draft [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. The FETCH method defined in the present 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.

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:

- o 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.
- o 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.

1.2. PATCH and iPATCH

PATCH is also specified for HTTP in [RFC5789]. Most of the motivation for PATCH described in [RFC5789] also applies here. iPATCH is the idempotent version of PATCH.

The PUT method exists to overwrite a resource with completely new contents, and cannot be used to perform partial changes. When using PUT for partial changes, proxies and caches, and even clients and servers, may get confused as to the result of the operation. PATCH was not adopted in an early design stage of CoAP, however, it has become necessary with the arrival of applications that require partial updates to resources (e.g. [I-D.vanderstok-core-comi]). Using PATCH avoids transferring all data associated with a resource in case of modifications, thereby not burdening the constrained communication medium.

This document relies on knowledge of the PATCH specification for HTTP [RFC5789]. This document provides extracts from [RFC5789] to make independent reading possible.

1.3. Requirements Language

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].

1.4. Terminology and Acronyms

This document uses terminology defined in [RFC5789] and [RFC7252].

2. FETCH Method

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 as described by the request parameters (including the request options and the 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.

Together with the request options, the body of the request (which may be constructed from multiple payloads using the block protocol [I-D.ietf-core-block]) 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 a 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 intended 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 a successful 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 body 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 cache 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.

TODO 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. FETCH 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. PATCH and iPATCH Methods

The PATCH and iPATCH methods request that a set of changes described in the request payload is applied to the target resource of the request. The set of changes is represented in a format identified by a media type. If the Request-URI does not point to an existing resource, the server MAY create a new resource with that URI, depending on the patch document type (whether it can logically modify a null resource) and permissions, etc. Creation of a new resource would result in a 2.01 (Created) Response Code dependent of the patch document type.

Restrictions to a PATCH or iPATCH request can be made by including the If-Match or If-None-Match options in the request (see Section 5.10.8.1 and 5.10.8.2 of [RFC7252]). If the resource could not be created or modified, then an appropriate Error Response Code SHOULD be sent.

The difference between the PUT and PATCH requests is extensively documented in [RFC5789].

The PATCH method is not safe and not idempotent, as with the HTTP PATCH method specified in [RFC5789].

The iPATCH method is not safe but idempotent, as with the CoAP PUT method specified in [RFC7252], Section 5.8.3.

A client can mark a request as idempotent by using the iPATCH method instead of the PATCH method. This is the only difference between the two. The indication of idempotence may enable the server to keep less state about the interaction; some constrained servers may only implement the iPATCH variant for this reason.

PATCH and iPATCH are both atomic. The server MUST apply the entire set of changes atomically and never provide a partially modified representation to a concurrently executed GET request. Given the constrained nature of the servers, most servers will only execute CoAP requests consecutively, thus preventing a concurrent partial overlapping of request modifications. Resuming, modifications MUST NOT be applied to the server state when an error occurs or only a partial execution is possible on the resources present in the server.

The atomicity applies to a single server. When a PATCH or iPATCH request is multicast to a set of servers, each server can either execute all required modifications or not. It is not required that all servers execute all modifications or none. An Atomic Commit protocol that provides multiple server atomicity is out of scope.

A PATCH or iPATCH response can invalidate a cache as with the PUT response. Caching behaviour as function of the successful (2.xx) response codes for PATCH or iPATCH are:

- o A 2.01 (Created) response invalidates any cache entry for the resource indicated by the Location-* Options; the payload is a representation of the action result.
- o A 2.04 (Changed) response invalidates any cache entry for the target resource; the payload is a representation of the action result.

There is no guarantee that a resource can be modified with PATCH or iPATCH. Servers MUST ensure that a received PATCH body is appropriate for the type of resource identified by the target resource of the request.

When a request is intended to effect a partial update of a given resource, clients cannot use PUT while supplying just the update, but are free to use PATCH or iPATCH.

3.1. Simple Examples for PATCH and iPATCH

The example is taken over from [RFC6902], which specifies a JSON notation for PATCH operations. A resource located at `coap://www.example.com/object` contains a target JSON document.

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

REQ: iPATCH CoAP://www.example.com/object
Content-Format: application/json-patch+json

```
[
  { "op": "replace", "path": "x-coord", "value": 45 }
]
```

RET: CoAP 2.04 Changed

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

This example illustrates use of an idempotent modification to the `x-coord` member of the existing resource `"object"`. The 2.04 (Changed) response code is conform with the CoAP PUT method.

The same example using the Content-Format `application/merge-patch+json` from [RFC7396] looks like:

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

```
REQ: iPATCH CoAP://www.example.com/object
Content-Format: 52 (application/merge-patch+json)
  { "x-coord":45}
```

RET: CoAP 2.04 Changed

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

The examples show the use of the iPATCH method, but the use of the PATCH method would have led to the same result. Below a non-idempotent modification is shown. Because the action is non-idempotent, iPATCH returns an error, while PATCH executes the action.

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

```
REQ: iPATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)
[
  { "op": "add", "path": "foo/1", "value": "bar" }
]
RET: CoAP 4.12 Precondition Failed
```

JSON document final state is unchanged

```
REQ: PATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)
[
  { "op": "add", "path": "foo/1", "value": "bar" }
]
RET: CoAP 2.04 Changed
```

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "bar", "baz"]
}
```

3.2. Response Codes

PATCH and iPATCH for CoAP adopt the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252].

3.3. Option Numbers

PATCH and iPATCH for CoAP adopt the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

3.4. Error Handling

A PATCH or iPATCH request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH or iPATCH payload: If a server determines that the payload provided with a PATCH or iPATCH request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The

definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported PATCH or iPATCH payload: In case a client sends payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a PATCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 4.22 (Unprocessable Entity) CoAP error. More specific scenarios might include situations when:

- * the server has insufficient computing resources to complete the request successfully -- 4.13 (Request Entity Too Large) CoAP Response Code (see below),
- * the resource specified in the request becomes invalid by applying the payload -- 4.09 (Conflict) CoAP Response Code (see below),

In case there are more specific errors that provide more insight into the problem, then those should be used.

Resource not found: The 4.04 (Not Found) error should be returned in case the payload of a PATCH request cannot be applied to a non-existent resource.

Failed precondition: In case the client uses the conditional If-Match or If-None-Match option to define a precondition for the PATCH request, and that precondition fails, then the server can return the 4.12 (Precondition Failed) CoAP error.

Request too large: If the payload of the PATCH request is larger than a CoAP server can process, then it can return the 4.13 (Request Entity Too Large) CoAP error.

Conflicting state: If the modification specified by a PATCH or iPATCH request causes the resource to enter an inconsistent state that the server cannot resolve, the server can return the 4.09 (Conflict) CoAP response. The server SHOULD generate a payload that includes enough information for a user to recognize the source of the conflict. The server MAY return the actual resource state to provide the client with the means to create a new consistent resource state. Such a situation might be encountered

when a structural modification is applied to a configuration data-store, but the structures being modified do not exist.

Concurrent modification: Resource constrained devices might need to process requests in the order they are received. In case requests are received concurrently to modify the same resource but they cannot be queued, the server can return a 5.03 (Service unavailable) CoAP response code.

Conflict handling failure: If the modification implies the reservation of resources or the waiting on conditions to become true, leading to a too long request execution time, the server can return 5.03 (service unavailable) response code.

It is possible that other error situations, not mentioned here, are encountered by a CoAP server while processing the PATCH request. In these situations other appropriate CoAP status codes can also be returned.

4. Discussion

Adding three new methods to CoAP's existing four may seem like a major change. However, both FETCH and the two PATCH variants fit well into the REST paradigm and have been anticipated on the HTTP side. Adding both a non-idempotent and an idempotent PATCH variant allows to keep interoperability with HTTP's PATCH method as well as the use/indication of an idempotent PATCH if that is possible, saving significant effort on the server side.

Interestingly, the three new methods fit into the old table of methods with a surprising similarity in the idempotence and safety attributes:

Code	Name	Code	Name	safe	idempotent
0.01	GET	0.05	FETCH	yes	yes
0.02	POST	0.06	PATCH	no	no
0.03	PUT	0.07	iPATCH	no	yes
0.04	DELETE			no	yes

5. Security Considerations

This section analyses the possible threats to the CoAP FETCH and PATCH or iPATCH methods. It is meant to inform protocol and application developers about the security limitations of CoAP FETCH and PATCH or iPATCH as described in this document.

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

The security consideration of section 15 of [RFC2616], section 11 of [RFC7252], and section 5 of [RFC5789] also apply.

The security considerations for PATCH or iPATCH are nearly identical to the security considerations for PUT ([RFC7252]). The mechanisms used for PUT can be used for PATCH or iPATCH as well.

PATCH or iPATCH are secured following the CoAP recommendations as specified in section 9 of [RFC7252]. When additional security techniques are standardized for CoAP, PATCH or iPATCH can also be (and need to be) secured by those new techniques.

6. IANA Considerations

IANA is requested to add the following entries to the sub-registry "CoAP Method Codes":

Code	Name	Reference
0.05	FETCH	[RFCthis]
0.06	PATCH	[RFCthis]
0.07	iPATCH	[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.

The PATCH method is neither idempotent nor safe. It returns the same response codes that POST can return, plus 4.09 "Conflict" with the semantics specified in Section 3.4.

IANA is requested to add the following code to the sub-registry "CoAP response codes":

Code	Name	Reference
4.09	Conflict	[RFCthis]

IANA is requested to add entries to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry:

Media Type	Encoding	ID	Reference
application/json-patch+json		51	[RFC6902]
application/merge-patch+json		52	[RFC7396]

7. Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP concerning PATCH, and found a number of problems in an earlier version of Section 2. We are grateful for discussions with Christian Amsuss, Timothy Carey, Paul Duffy, Kovatsch Matthias, Michel Veillette, Michael Verschoor, Thomas Watteyne, and Gengyu Wei.

8. Change log

When published as a RFC, this section needs to be removed.

Version 00 is a composition from draft-vanderstok-core-patch-03 and draft-bormann-core-coap-fetch-00 and replaces these two drafts.

9. References

9.1. 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, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <<http://www.rfc-editor.org/info/rfc2616>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <<http://www.rfc-editor.org/info/rfc5789>>.

- [RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013, <<http://www.rfc-editor.org/info/rfc6902>>.
- [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>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <<http://www.rfc-editor.org/info/rfc7396>>.
- [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>>.
- [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.

9.2. Informative References

- [RFC5323] Reschke, J., Ed., Reddy, S., Davis, J., and A. Babich, "Web Distributed Authoring and Versioning (WebDAV) SEARCH", RFC 5323, DOI 10.17487/RFC5323, November 2008, <<http://www.rfc-editor.org/info/rfc5323>>.
- [I-D.vanderstok-core-comi]
Stok, P. and A. Bierman, "CoAP Management Interface", draft-vanderstok-core-comi-09 (work in progress), March 2016.
- [I-D.hartke-core-apps]
Hartke, K., "CoRE Application Descriptions", draft-hartke-core-apps-03 (work in progress), February 2016.
- [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.

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

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

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

Anuj Sehgal
Consultant

Email: anuj@iurs.org