Internet Engineering Task Force (IETF)
Request for Comments: 8132
Category: Standards Track
ISSN: 2070-1721

P. van der Stok
Consultant
C. Bormann
Universitaet Bremen TZI
A. Sehgal
NAVOMI, Inc.

April 2017

PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)

Abstract

The methods defined in RFC 7252 for the Constrained Application Protocol (CoAP) only allow access to a complete resource, not to parts of a resource. In case of resources with larger or complex data, or in situations where resource continuity is required, replacing or requesting the whole resource is undesirable. Several applications using CoAP need to access parts of the resources.

This specification defines the new CoAP methods, FETCH, PATCH, and iPATCH, which are used to access and update parts of a resource.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc8132.

Table of Contents

1. Introduction

Similar to HTTP, the GET method defined in [RFC7252] for the Constrained Application Protocol (CoAP) 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 some applications to use POST where a cacheable, idempotent, safe request is actually desired.

Again, similar to the original specification of HTTP, the PUT method defined in [RFC7252] only allows a complete resource to be replaced. This also leads applications to use POST where a cacheable, possibly idempotent request is actually desired.

The present 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 a CoAP 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 also shape the representation. This has been modeled 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 so 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. [HTTP-SEARCH] proposes a SEARCH method that is similar to GET in most properties but enables sending a request body, as is done with POST. The FETCH method defined in the present specification is inspired by [HTTP-SEARCH], 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:

- The client can specify a media type (and a content coding) that enables 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, which enables 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.

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., [COAP-MGMNT]). 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].

Specifically, it uses the terms "safe" and "idempotent" as defined in Section 5.1 of [RFC7252]. (Further discussion of safe and idempotent methods can now be found in Sections 4.2.1 and 4.2.2 of [RFC7231], respectively; the implications of idempotence of methods on server implementations are also discussed in Section 4.5 of [RFC7252].)

2. FETCH Method

The CoAP FETCH method is used to obtain a representation of a resource, specified by 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, i.e., it cannot be used by a cache as a payload to be returned by a GET request.

Together with the request options, the body of the request (which may be constructed from multiple payloads using the block protocol [RFC7959]) defines the request parameters. With the FETCH method, implementations may submit a request body of any media type that is defined with the semantics of selecting information from a resource in such a FETCH request; it is outside the scope of this document how information about media types admissible for the specific resource is obtained by the client (although we can hint that form relations [CORE-APP] might be a preferred way). It is RECOMMENDED that any discovery method that allows a client to find out that the server supports FETCH also provides information regarding what FETCH payload media types are applicable.

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 FETCH 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 typical way to respond to a FETCH request. (Note that this aspect differs markedly from [HTTP-SEARCH] and 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.

2.1. Response Codes

FETCH for CoAP adopts the response codes as specified in Sections 5.9 and 12.1.2 of [RFC7252] as well as the additional response codes mentioned in Section 2.2.

2.2. Error Handling

A FETCH request may fail under certain known conditions. Beyond the conditions already defined in [RFC7252] for GET, noteworthy ones are:

Malformed FETCH payload: If a server determines that the payload provided with a FETCH 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 FETCH payload: In case a client sends a 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 FETCH request is determined to be valid (i.e., well-formed and supported) but the server is unable to or is incapable of processing the request. The server can return a 4.22 (Unprocessable Entity) CoAP error. In situations when the server has insufficient computing resources to complete the request successfully, it can return a 4.13 (Request Entity Too Large) CoAP error (see also below). If there are more specific errors that provide additional insight into the problem, then those should be used.

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

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

2.3. Option Numbers

FETCH for CoAP adopts the option numbers as specified in Sections 5.10 and 12.2 of [RFC7252].

Generally, options defined for GET act in an analogous way for FETCH. Two specific cases are called out in the rest of this section.

2.3.1. The Content-Format Option

A FETCH request MUST include a Content-Format option (see Section 5.10.3 of [RFC7252]) to specify the media type and content coding of the request body. (Typically, the media type will have been specifically designed to specify details for a selection or a search on a resource.)

2.3.2. The ETag Option

The ETag option on a FETCH result has the same semantics as defined in Section 5.10.6 of [RFC7252]. In particular, its use as a response option describes the "tagged representation", which for FETCH is the same as the "selected representation". The FETCH payload is input to that selection process and therefore needs to be part of the cache key. Similarly, the use of ETag as a request option can elicit a 2.03 (Valid) response if the representation associated with the ETag would still be selected by the FETCH request (including its payload).

2.4. Working with Observe

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

2.5. Working with Block

The Block1 option [RFC7959] can be used with a FETCH request as it would be used with a POST request; the Block2 option can then be used as it would with GET or POST.

2.6. Building FETCH Requests

One property of FETCH that may be non-obvious is that a FETCH request cannot be generated from a link alone; the client also needs a way to generate the request payload. Again, form relations [CORE-APP] may be able to fill parts of this gap.

2.7. A Simple Example for FETCH

The FETCH method needs a media type for its payload (as expressed by the Content-Format request option) that specifies the search query in similar detail as is shown for the PATCH payload in the PATCH example in Section 3.1. ([HTTP-SEARCH] invents a "text/query" format based on some hypothetical SQL dialect for its examples.)

The example below illustrates retrieval of a subset of a JSON [RFC7159] object (the same object as used in Section 3.1). Using a hypothetical media type "application/example-map-keys+json" (with a Content-Format ID of NNN, which is not defined as this is just an example), the client specifies the items in the object that it wants: it supplies a JSON array that gives the map keys for these items. A resource located at <coap://www.example.com/object> can be represented by a JSON document that we will consider as the target of the FETCH. The client wants to learn the contents of the single map key "foo" within this target:

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

FETCH Example: JSON Document Returned by GET

The example FETCH request specifies a single top-level member desired by giving its map key as the sole element of the "example-map-keys" payload:

```
FETCH CoAP://www.example.com/object
Content-Format: NNN (application/example-map-keys+json)
Accept: application/json
[
  "foo"
]
```

FETCH Example: Request

The server returns a subset document with just the selected member:

```
2.05 Content
Content-Format: 50 (application/json)
{
  "foo": ["bar","baz"]
}
```

FETCH Example: Response with Subset JSON Document

By the logic of this example, the requester could have entered more than one map key into the request payload array and would have received a more complete subset of the top-level JSON object that is representing the resource.

3. PATCH and iPATCH Methods

The PATCH and iPATCH methods request that a set of changes described in the request payload be 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, as well as other conditions such as the degree of control the server gives clients in creating new

entries in its URI space (see also Section 3.4). Creation of a new resource would result in a 2.01 (Created) response code dependent on 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 Sections 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 documented in [RFC5789]. When a request is intended to effect a partial update of a given resource, clients cannot use PUT while supplying just the update, but they might be able to use PATCH or iPATCH.

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

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

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. In other words, modifications MUST NOT be applied to the server state when an error occurs or when 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 in a similar manner to the PUT response. For the successful (2.xx) response codes, PATCH or iPATCH have the following caching behavior:

- 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.

- 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.

It is RECOMMENDED that any discovery method that allows a client to find out that the server supports one of PATCH and iPATCH also provide information regarding what PATCH payload media types are applicable and which of the two methods are implemented by the server for each of these media types.

Servers that do not rely on the idempotence of iPATCH can easily support both PATCH and iPATCH, and it is RECOMMENDED they do so. This is inexpensive to do, as, for iPATCH, there is no requirement on the server to check that the client’s intention that the request be idempotent is fulfilled (although there is diagnostic value in that check, so a less-constrained implementation may want to perform it).

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: 51 (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 conforms with the CoAP PUT method.

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

```
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 4.00 Bad Request
Diagnostic payload: Patch format not idempotent

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] and add 4.09 (Conflict) and 4.22 (Unprocessable Entity) with the semantics specified in Section 3.4 of the present specification.

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 a 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 to be valid (i.e., well-formed and supported) but the server is unable to or is incapable of processing the request. The server can return a 4.22 (Unprocessable Entity) CoAP error. More specific scenarios might include situations such as:

* the server has insufficient computing resources to complete the request successfully -- 4.13 (Request Entity Too Large) CoAP response code (see below); or

* the resource specified in the request becomes invalid by applying the payload -- 4.09 (Conflict) CoAP response code (see "Conflicting state" below)).

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

Resource not found: The 4.04 (Not Found) error should be returned if 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 wait time for conditions to become true leads to a too-long request execution time, the server can return a 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. The New Set of CoAP Methods

Adding three new methods to CoAP's existing four may seem like a major change. However, 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 interoperability with HTTP's PATCH method to be kept and allows the use/indication of an idempotent PATCH when that is possible, which saves 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 analyzes 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 Section 11 of [RFC7252]. Specifically, the security considerations for FETCH are closest to those of GET, except that the FETCH request carries a payload that may need additional protection. The payload of a FETCH request may reveal more detailed information about the specific portions of a resource of interest to the requester than a GET request for the entire resource would; this may mean that confidentiality protection of the request by Datagram Transport Layer Security (DTLS) or other means is needed for FETCH where it wouldn’t be needed for GET.

The PATCH and iPATCH methods are subject to the same general security considerations as all CoAP methods as described in Section 11 of [RFC7252]. The specific security considerations for PATCH or iPATCH are nearly identical to the security considerations for PUT [RFC7252]; the security considerations of Section 5 of [RFC5789] also apply to PATCH and iPATCH. Specifically, there is likely to be a need for authorizing requests (possibly through access control and/or authentication) and for ensuring that data is not corrupted through transport errors or through accidental overwrites. The mechanisms used for PUT can be used for PATCH or iPATCH as well.

The new methods defined in the present specification are secured following the CoAP recommendations for the existing methods as specified in Section 9 of [RFC7252]. When additional security techniques are standardized for CoAP (e.g., Object Security), these techniques are then also available for securing the new methods.

6. IANA Considerations

IANA has added the following entries to the subregistry "CoAP Method Codes":

Code	Name	Reference
0.05	FETCH	RFC 8132
0.06	PATCH	RFC 8132
0.07	iPATCH	RFC 8132

The FETCH method is idempotent and safe, and it returns the same response codes that GET can return, plus 4.13 (Request Entity Too Large), 4.15 (Unsupported Content-Format), and 4.22 (Unprocessable Entity) with the semantics specified in Section 2.2.

The PATCH method is neither idempotent nor safe. It returns the same response codes that POST can return, plus 4.09 (Conflict) and 4.22 (Unprocessable Entity) with the semantics specified in Section 3.4.

The iPATCH method is identical to the PATCH method, except that it is idempotent.

IANA has added the following code to the subregistry "CoAP Response Codes":

Code	Name	Reference
4.09	Conflict	RFC 8132
4.22	Unprocessable Entity	RFC 8132

IANA has added entries to the subregistry "CoAP Content-Formats":

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

7. References

7.1. 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>.

[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>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

[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>.

[RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <http://www.rfc-editor.org/info/rfc7959>.

7.2. 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>.

[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>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <http://www.rfc-editor.org/info/rfc7396>.

[COAP-MGMNT] Stok, P., Bierman, A., Veillette, M., and A. Pelov, "CoAP Management Interface", Work in Progress, draft-ietf-core-comi-00, January 2017.

[CORE-APP] Hartke, K., "CoRE Application Descriptions", Work in Progress, draft-hartke-core-apps-07, February 2017.

[HTTP-SEARCH] Reschke, J., Malhotra, A., and J. Snell, "HTTP SEARCH Method", Work in Progress, draft-snell-search-method-00, April 2015.

Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP concerning PATCH and found a number of problems in an earlier draft version of Section 2. We are grateful for discussions with Christian Amsuss, Andy Bierman, Timothy Carey, Paul Duffy, Matthias Kovatsch, Michel Veillette, Michael Verschoor, Thomas Watteyne, and Gengyu Wei. Christian Groves provided detailed comments during the Working Group Last Call, and Christer Holmberg’s Gen-ART review provided some further editorial improvement. Further Last Call reviews were provided by Sheng Jiang and Phillip Hallam-Baker. As usual, the IESG had some very good reviews, and we would like to specifically call out those by Alexey Melnikov (responsible AD) and Alissa Cooper.

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
NAVOMI, Inc.

Email: anuj.sehgal@navomi.com