Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: May 29, 2010

L. Dusseault
Linden Lab
J. Snell

November 25, 2009

PATCH Method for HTTP
draft-dusseault-http-patch-16

Abstract

Several applications extending the Hypertext Transfer Protocol (HTTP) require a feature to do partial resource modification. The existing HTTP PUT method only allows a complete replacement of a document. This proposal adds a new HTTP method, PATCH, to modify an existing HTTP resource.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on May 29, 2010.

Copyright Notice

Table of Contents

1. Introduction

This specification defines the new HTTP/1.1 [RFC2616] method PATCH that is used to apply partial modifications to a resource.

A new method is necessary to improve interoperability and prevent errors. The PUT method is already defined to overwrite a resource with a complete new body, and can not be reused to do partial changes. Otherwise, proxies and caches and even clients and servers may get confused as to the result of the operation. POST is already used but without broad interoperability (for one, there is no standard way to discover patch format support). PATCH was mentioned in earlier HTTP specifications, but not completely defined.

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in [RFC2119].

Furthermore, this document uses the ABNF syntax defined in Section 2.1 of [RFC2616].

2. The PATCH Method

The PATCH method requests that a set of changes described in the request entity be applied to the resource identified by the Request-URI. The set of changes is represented in a format called a "patch document" identified by a media type. If the Request-URI does not point to an existing resource, the server MAY create a new resource, depending on the patch document type (whether it can logically modify a null resource) and permissions etc.

The difference between the PUT and PATCH requests is reflected in the way the server processes the enclosed entity to modify the resource identified by the Request-URI. In a PUT request, the enclosed entity is considered to be a modified version of the resource stored on the

origin server and the client is requesting that the stored version be replaced. With PATCH, however, the enclosed entity contains a set of instructions describing how a resource currently residing on the origin server should be modified to produce a new version. The PATCH method affects the resource identified by the Request-URI, and also MAY have side effects on other resources; i.e., new resources may be created, or existing ones modified, by the application of a PATCH.

PATCH is neither safe or idempotent as defined by [RFC2616], Section 9.1.

A PATCH request can be issued in such a way as to be idempotent,

which also helps prevent bad outcomes from collisions between two PATCH requests on the same resource in a similar timeframe. Collisions from multiple PATCH requests may be more dangerous than PUT collisions, because some patch formats need to operate from a known base point or else corrupt the resource. Clients using this kind of patch application SHOULD acquire a strong ETag [RFC2616] for the resource to be modified, and use that ETag in the If-Match header on the PATCH request to verify that the resource is still unchanged. If a strong ETag is not available for a given resource, the client can use If-Unmodified-Since as a less-reliable safeguard.

There are also cases where patch formats do not need to operate from a known base-point (e.g. appending text lines to log files, or non-colliding rows to database tables), in which case the same care in client requests is not needed.

The server MUST apply the entire set of changes atomically and never provide (e.g. in response to a GET during this operation) a partially-modified representation. If the entire patch document cannot be successfully applied then the server MUST NOT apply any of the changes. The determination of what constitutes a successful PATCH can vary depending on the patch document and the type of resource(s) being modified. For example, the common 'diff' utility can generate a patch document that applies to multiple files in a directory hierarchy. The atomicity requirement holds for all directly affected files. See Error Handling in Section 2.2 for details on status codes and possible error conditions.

If the request passes through a cache and the Request-URI identifies

one or more currently cached entities, those entries SHOULD be treated as stale. A response to this method is only cacheable if it contains explicit freshness information (such as an Expires header or "Cache-Control: max-age" directive) as well as the Content-Location header matching the request-URI, indicating that the PATCH response body is a resource representation. A cached PATCH response can only be used to respond to subsequent GET and HEAD requests; it MUST NOT be used to respond to other methods (in particular, PATCH).

Note that entity-headers contained in the request apply only to the contained patch document and MUST NOT be applied to the resource being modified. Thus, a Content-Language header could be present on the request but it would only mean (for whatever that's worth) that the patch document had a language. Servers SHOULD NOT store such headers except as trace information, and SHOULD NOT use such header values the same way they might be used on PUT requests. Therefore, this document does not specify a way to modify a document's Content-Type or Content-Language value through headers, though a mechanism could well be designed to achieve this goal through a patch document.

There is no guarantee that a resource can be modified with PATCH. Further, it is expected that different patch document formats will be appropriate for different types of resources and that no single format will be appropriate for all types of resources. Therefore, there is no single default patch document format that implementations are required to support. Servers MUST ensure that a received patch document is appropriate for the type of resource identified by the Request-URI.

Clients need to choose when to use PATCH rather than PUT. For example, if the patch document size is larger than the size of the new resource data that would be used in a PUT, then it might make sense to use PUT instead of PATCH. A comparison to POST is even more difficult, because POST is used in widely varying ways and can encompass PUT and PATCH-like operations if the server chooses. If the operation does not modify the resource identified by the Request-URI in a predictable way, POST should be considered instead of PATCH or PUT.

2.1. A simple PATCH example

```
PATCH /file.txt HTTP/1.1
```

```
Host: www.example.com
Content-Type: application/example
If-Match: "e0023aa4e"
Content-Length: 100

[description of changes]
```

This example illustrates use of a hypothetical patch document on an existing resource. The 204 response code is used because the response does not have a body (a response with the 200 code would have a body) but other success codes can be used if appropriate.

Successful PATCH response to existing text file

```
HTTP/1.1 204 No Content
Content-Location: /file.txt
ETag: "e0023aa4f"
```

2.2. Error handling

There are several known conditions under which a PATCH request can fail.

Malformed patch document: When the server determines that the patch document provided by the client is not properly formatted, it SHOULD return a 400 (Bad Request) response. The definition of badly formatted depends on the patch document chosen.

Unsupported patch document: Can be specified using a 415 (Unsupported Media Type) when the client sends a patch document format that the server does not support for the resource identified by the Request-URI. Such a response SHOULD include an Accept-Patch response header as described in Section 3.1 to notify the client what patch document media types are supported.

Unprocessable request: Can be specified with a 422 (Unprocessable Entity) ([RFC4918], Section 11.2) when the server understands the patch document and the syntax of the patch document appears valid, but the server is incapable of processing the request. This might include attempts to modify a resource in a way that would cause

the resource to become invalid: for instance, a modification to a well-formed XML document that would cause it to no longer be well-formed. There may also be more specific errors like "Conflicting State" that could be signaled with this status code, but the more specific error would generally be more helpful.

Resource Not Found: Can be specified with a 404 (Not Found) status code, when the client attempted to apply a patch document to a non-existent resource, but the patch document chosen cannot be applied to a non-existent resource.

Conflicting State: Can be specified with a 409 (Conflict) when the request cannot be applied given the state of the resource. For example, if the client attempted to apply a structural modification and the structures assumed to exist did not exist (with XML, a patch might specify changing element 'foo' to element 'bar' but element 'foo' might not exist).

Conflicting modification: When a client uses either the If-Match or If-Unmodified-Since header to define a precondition, and that precondition failed, then the 412 (Precondition Failed) error is most helpful to the client. However, that response makes no sense if there was no precondition on the request. In cases when the server detects a possible conflicting modification and no precondition was defined in the request, the server can return a 409 (Conflict) response.

Concurrent modification: Some applications of PATCH might require the server to process requests in the order in which they are received. If a server is operating under those restrictions, and it receives concurrent requests to modify the same resource, but is unable to queue those requests, the server can usefully indicate this error by using a 409 (Conflict) response.

Note that the 409 Conflict response gives reasonably consistent information to clients. Depending on the application and the nature of the patch format, the client might be able to reissue the request

as is (e.g. an instruction to append a line to a log file), or it might have to retrieve the resource content to recalculate a patch, or it might have to fail the operation.

Other HTTP status codes can also be used under the appropriate circumstances.

The entity body of error responses SHOULD contain enough information

to communicate the nature of the error to the client. The content-type of the response entity can vary across implementations.

3. Advertising Support in OPTIONS

A server can advertise its support for the PATCH method by adding it to the listing of allowed methods in the "Allow" OPTIONS response header defined in HTTP/1.1. The PATCH method MAY appear in the "Allow" header even if the Accept-Patch header is absent, in which case the list of allowed patch documents is not advertised.

3.1. The Accept-Patch Header

This specification introduces a new response header "Accept-Patch" used to specify the patch document formats accepted by the server. "Accept-Patch" SHOULD appear in the OPTIONS response for any resource that supports the use of the PATCH method. The presence of the "Accept-Patch" header in response to any method is an implicit indication that PATCH is allowed on the resource identified by the Request-URI. The presence of a specific patch document format in this header indicates that specific format is allowed on the resource identified by the Request-URI.

Accept-Patch = "Accept-Patch" ":" 1#media-type

The Accept-Patch header specifies a comma separated listing of media-types as defined by [RFC2616], Section 3.7.

3.2. Example OPTIONS Request and Response

[request]

```
OPTIONS /example/buddies.xml HTTP/1.1
Host: www.example.com
```


[response]

```
HTTP/1.1 200 OK
Allow: GET, PUT, POST, OPTIONS, HEAD, DELETE, PATCH
Accept-Patch: application/example, text/example
```

The examples show a server that supports PATCH generally using two hypothetical patch document formats.

4. IANA Considerations

4.1. The 'Accept-Patch' Response Header

The 'Accept-Patch' response header should be added to the permanent registry (see [RFC3864]).

```
Header field name:  Accept-Patch
Applicable Protocol:  HTTP
Author/Change controller:  IETF
Specification document:  this specification
```

5. Security Considerations

The security considerations for PATCH are nearly identical to the security considerations for PUT ([RFC2616], Section 9.6). These include authorizing requests (possibly through access control and/or authentication) and ensuring that data is not corrupted through transport errors or through accidental overwrites. Whatever mechanisms are used for PUT can be used for PATCH as well. The following considerations apply specially to PATCH.

A document that is patched might be more likely to be corrupted than a document that is overridden in entirety, but that concern can be addressed through the use of mechanisms such as conditional requests using ETags and the If-Match request header as described in Section 2. If a PATCH request fails, the client can issue a GET request to the resource to see what state it is in. In some cases, the client might be able to check the contents of the resource to see if the PATCH request can be resent, but in other cases the attempt will just fail and/or a user will have to verify intent. In the case of a failure of the underlying transport channel, where a PATCH response is not received before the channel fails or some other timeout happens, the client might have to issue a GET request to see whether the request was applied. The client might want to ensure that the GET request bypasses caches using mechanisms described in HTTP specifications (see for example Section 13.1.6 of [RFC2616]).

Sometimes an HTTP intermediary might try to detect viruses being sent via HTTP by checking the body of the PUT/POST request or GET response. The PATCH method complicates such watch-keeping because neither the source document nor the patch document might be a virus, yet the result could be. This security consideration is not materially different from those already introduced by byte-range downloads, downloading patch documents, uploading zipped (compressed) files and so on.

Individual patch documents will have their own specific security considerations that will likely vary depending on the types of resources being patched. The considerations for patched binary resources, for instance, will be different than those for patched XML documents. Servers MUST take adequate precautions to ensure that malicious clients cannot consume excessive server resources (e.g., CPU, disk I/O) through the client's use of PATCH.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[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, June 1999.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

6.2. Informative References

[RFC4918] Dusseault, L., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

Appendix A. Acknowledgements

PATCH is not a new concept, it first appeared in HTTP in drafts of version 1.1 written by Roy Fielding and Henrik Frystyk and also appears in Section 19.6.1.1 of RFC 2068.

Thanks to Adam Roach, Chris Sharp, Julian Reschke, Geoff Clemm, Scott

Lawrence, Jeffrey Mogul, Roy Fielding, Greg Stein, Jim Luther, Alex Rousskov, Jamie Lokier, Joe Hildebrand, Mark Nottingham, Michael

Balloni, Cyrus Daboo, Brian Carpenter, John Klensin, Eliot Lear and SM for review and advice on this document.

Appendix B. Changes

B.1. Changes from -00

OPTIONS support: removed "Patch" header definition and used Allow and new "Accept-Patch" headers instead.

Supported delta encodings: removed vcdiff and diffe as these do not have defined MIME types and did not seem to be strongly desired.

PATCH method definition: Clarified cache behavior.

B.2. Changes from -01

Removed references to XCAP - not yet a RFC.

Fixed use of MIME types (this "fix" now obsolete)

Explained how to use MOVE or COPY in conjunction with PATCH, to create a new resource based on an existing resource in a different location.

B.3. Changes from -02

Clarified that MOVE and COPY are really independent of PATCH.

Clarified when an ETag must change, and when Last-Modified must be used.

Clarified what server should do if both Content-Type and IM headers appear in PATCH request.

Filled in missing reference to DeltaV and ACL RFCs.

Stopped using 501 Unsupported for unsupported delta encodings.

Clarified what a static resource is.

Refixed use of MIME types for patch formats.

Limited the scope of some restrictions to apply only to usage of required diff format.

B.4. Changes from -03

Various typographical, terminology consistency, and other minor clarifications or fixes.

B.5. Changes from -04

Moved paragraphs on ACL and RFC3229 interoperability to new section.

Added security considerations.

Added IANA considerations, registration of new namespace, and discontinued use of "DAV:" namespace for new elements.

Added example of error response.

B.6. Changes from -05

Due to various concerns it didn't seem likely the application/gdiff registration could go through so switching to vcdiff as required diff format, and to RFC3229's approach to specifying diff formats, including use of the IM header.

Clarified what header server MUST use to return MD5 hash.

Reverted to using 501 Unsupported for unsupported delta encodings.

B.7. Changes from -06

The reliance on RFC 3229 defined patch documents has been factored out in favor of delta encodings identified by MIME media type.

The required use of DeltaV-based error reporting has been removed in favor of using basic HTTP status codes to report error conditions.

The Accept-Patch response header has been redefined as a listing of media-ranges, similar to the Accept request header.

Added James Snell as a co-author.

B.8. Changes from -07

Terminology change from "delta encoding" to "patch document"

Added clarification on the safety and idempotency of PATCH

Updated the caching rules of PATCH responses

200 responses MUST include a representation of the modified resource. 204 responses are used to indicate successful response without returning a representation.

Suggest using 422 Unprocessable Entity to indicate that a properly formatted patch document cannot be processed

Clarify the use of 412 and 409 to indicate concurrent and conflicting resource modifications.

Added registration for the Accept-Patch header.

Relaxed the requirements for the use of If-Match and If-Unmodified-Since.

Add language that clarifies the difference between PUT and PATCH.

Add language that clarifies the issues with PATCH and Content Negotiation.

Use of Accept-Patch on any response implies that PATCH is supported.

Add language advising caution when pipelining PATCH requests.

B.9. Changes from -08

Addition of the 209 Content Returned status code

Addition of the Prefer header field mechanism

Removed the paragraph discussing the use of 200+Content-Location. This is replaced by the 209 Content Returned status code.

B.10. Changes from -09

Move the prefer header to a separate document

Restructure the document sections.

B.11. Changes from -10

Remove paragraph about pipelined requests. This is covered adequately by RFC2616.

Remove paragraph about content negotiation. This is covered adequately by RFC2616.

Explicitly indicate that PATCH can be used to create new resources.

Remove recommendation for servers to provide strong etags. This is recommendation is implied and does not need to be explicitly.

Change Allow-Patch to a listing of media-type and not media-range.

B.12. Changes from -11

Fix section links.

State that this uses RFC2616-style ABNF.

Fix grammar for Accept-Patch.

Remove requirements for handling entity-headers on PATCH and replace with general discussion of issues and consequences of having no handling requirements.

Update Security Considerations to make it clear what security

considerations for PUT are, for comparison.

B.13. Changes from -12

Remove status 209 again.

Add security consideration about using too much server resources.

Remove Content-MD5 from example.

B.14. Changes from -13

Remove '*' value from Accept-Patch again.

Allow caching but only if context is clear.

Clarify how some patch formats might allow creating a new document.

Add comparison of PATCH to POST

B.15. Changes from -14

Clarified that Accept-Patch header SHOULD appear in OPTIONS response -- it is not absolutely required

Clarified how server can indicate that a PATCH response body is cachable as a resource representation.

Removed suggestion that PATCH side-effects might be specified in the patch document specification -- this implied that side-effects could

exclusively be determined that way, but in fact side-effects are often determined by the server unilaterally.

B.16. Changes from -15

Clarifications on how conflicting PATCH requests can be avoided, and why not all use cases necessarily involve conflict

Added Content-Location to example response, so the ETag would be legit

Expanded security considerations on avoiding collisions, recovering from possible (unknown) collisions

Very slight reordering of paragraphs in section 2, for better flow

Clarified that the concurrent-modification status response is optional for servers, and explained what clients can do with that response

Updated text describing conflicting modifications: when 412 is used, vs 409

Appendix C. Notes to RFC Editor

The RFC Editor should remove this section and the Changes section.

Authors' Addresses

Lisa Dusseault
Linden Lab
945 Battery Street
San Francisco, CA 94111
USA

Email: lisa.dusseault@gmail.com

James M. Snell

Email: jasnell@gmail.com
URI: http://www.snellspace.com