HTTP
Internet-Draft
Intended status: Experimental
Expires: June 7, 2020

A. Wright
December 05, 2019

Partial Uploads in HTTP
draft-wright-http-partial-upload-01

Abstract

This document specifies a new media type intended for use in PATCH payloads that allows a resource to be uploaded in several segments, instead of a single large request.

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 https://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 June 7, 2020.

Copyright Notice

Copyright (c) 2019 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 (https://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

This introduces a mechanism that allows user agents to upload a document over several requests. Similar solutions have been known as partial uploads, segmented uploading, or resumable uploads.

HTTP is a stateless protocol, which implies that if a request is interrupted, there can be no way to resume it. This is not normally an issue if there is an alternate way of arriving to the desired state from an incomplete state transition. For example, if a download is interrupted, the user-agent may request just the missing parts in a Range request. However, if an upload is interrupted, no method exists for the client to synchronize its state with the server and only upload the remaining data; the entire request must be canceled and retried. This document standardizes a media type for PATCH and a new status code for uploading new resources over several segmented requests.

1.1. Notational Conventions

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.

This document uses ABNF as defined in [RFC5234] and imports grammar rules from [RFC7230].

For brevity, example HTTP requests or responses may add newlines or whitespace, or omit some headers necessary for message transfer.

2. Modifying a content range with PATCH

The PATCH method [RFC5789] allows a client to modify a resource in a specific way, as specified by the request payload. This document formalizes the concept of using "multipart/byteranges" [RFC7233] as a patch file, allowing usage in PATCH; and introduces a simplified form "message/byterange" that only patches a single range.

The "message/byterange" form may be used in a request as so:

```
PATCH /uploads/foo HTTP/1.1
Content-Type: message/byterange
Content-Length: 283
If-Match: "xyzzy"
If-Unmodified-Since: Sat, 29 Oct 1994 19:43:31 GMT

Content-Range: bytes 100-299/600
Content-Type: text/plain
Content-Length: 200

[200 bytes...]
```

This request asks to modify a 600-byte document, overwriting 200 bytes of it, starting at a 100-byte offset.

3. Segmented upload with PATCH

As an alternative to using PUT to create a new resource, the contents of a resource may be uploaded in segments, each written across several PATCH requests.

The first PATCH request creates the resource and uploads the first segment. To ensure the resource does not exist, the request SHOULD include "If-None-Match: *". The request payload is a "message/byterange" document containing the first segment of the resource to be uploaded, and the total length of the resource to be uploaded. Upon processing, the server returns "2__ Sparse Resource" indicating the document is error-free up to this point, but that more writes are necessary before the resource will be considered fully written.

Additional segments are uploaded with the same format.

When the final segment is uploaded, the server detects the resource is completely uploaded, and returns the final status code.

If the client loses the state of the upload, or the connection is terminated, the user agent can re-synchronize by issuing a "HEAD" request for the resource to get the current uploaded length. The

response will typically be 200 (OK) or 2__ (Sparse Resource). If 2__, the user agent may resume uploading the document from that offset.

3.1. Example

A single PUT request that creates a new file can be split apart into multiple PATCH requests. Here is an example that uploads a 600-byte document across three 200-byte segments.

The first PATCH request creates the resource:

```
PATCH /uploads/foo HTTP/1.1
Content-Type: message/byterange
Content-Length: 281
If-None-Match: *

Content-Range: bytes 0-199/600
Content-Type: text/plain
Content-Length: 200

[200 bytes...]
```

This request allocates a 600 byte document, and uploading the first 200 bytes of it. The server responds with 2__ (Sparse Resource), indicating that the resource has been allocated and all uploaded data is saved, but acknowledging the more data must still be uploaded by the client.

Additional requests upload the remainder of the document:

```
PATCH /uploads/foo HTTP/1.1
Content-Type: message/byterange
Content-Length: 283
If-None-Match: *

Content-Range: bytes 200-399/600
Content-Type: text/plain
Content-Length: 200

[200 bytes...]
```

This second request also returns 2__ (Sparse Resource), since there are still 200 bytes that are not written to.

A third request uploads the final portion of the document:

```
PATCH /uploads/foo HTTP/1.1
Content-Type: message/byterange
Content-Length: 283
If-None-Match: *

Content-Range: bytes 200-399/600
Content-Type: text/plain
Content-Length: 200

[200 bytes...]
```

Since the document is fully written to, the server responds with 200 (OK), the same response as if the entire 600 bytes were written in a PUT request.

4. Registrations

4.1. 2__ (Sparse Resource) status code

The 2__ (Sparse Resource) status code indicates that while the request succeeded, the request target is not ready for use, and the server is awaiting more data to be written.

In response to a GET request, representations returned with this status code might not be valid according to their media type, but could become valid once more data is appended.

In response to a PATCH request, it means the operation succeeded, but more uploads are necessary before the server can do anything else with the resource.

This is a 2xx class status because it indicates the request was filled as requested, and may safely be handled the same as a 200 (OK) response. However, it is only expected to be seen by clients making partial uploads; clients not expecting this status MAY treat it as an error.

Responses to a HEAD request MUST return the same end-to-end headers as a GET request. Normally, HTTP allows HEAD responses to omit certain header fields related to the payload; however Content-Length and Content-Range are essential fields for synchronizing the state of partial uploads. Hop-by-hop headers may still be omitted.

Several alternate names for this status code can be considered, including: Incomplete Content, Partial Resource, or Incomplete Upload.

4.2. message/byterange media type

The "message/byterange" media type patches the defined byte range to some specified contents. It is similar to the "multipart/byteranges" media type, except it omits the multipart separator, and so only allows a single range to be specified.

It follows the syntax of HTTP message headers and body. It MUST include the Content-Range header field. If the message length is known by the sender, it SHOULD contain the Content-Length header field. Unknown or nonapplicable header fields MUST be ignored.

"header-field" and "message-body" are specified in [RFC7230].

```
byterange-document = *( header-field CRLF )
                     CRLF
                     [ message-body ]
```

A patch is applied to a document by changing the range of bytes to the contents of the patch message payload. Servers MAY treat an invalid or nonexistent range as an error.

5. Security Considerations

5.1. Unallocated ranges

Servers must consider what happens when clients make writes to a sparse file.

Servers will normally only allow patch ranges to start inside or immediately after the end of the representation. Servers supporting sparse writes MUST NOT disclose the contents of memory. This may be done at file creation time, or left to the filesystem if it can guarantee this behavior.

6. References

6.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, DOI 10.17487/RFC7233, June 2014, <https://www.rfc-editor.org/info/rfc7233>.

[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>.

6.2. Informative References

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <https://www.rfc-editor.org/info/rfc5789>.

Author's Address

Austin Wright

Email: aaa@bzfx.net