Workgroup: HTTP
Internet-Draft:
draft-ietf-httpbis-resumable-upload-03
Published: 4 March 2024
Intended Status: Standards Track
Expires: 5 September 2024
Authors: M. Kleidl, Ed. Transloadit
G. Zhang, Ed. Apple Inc.
L. Pardue, Ed. Cloudflare

Resumable Uploads for HTTP

Abstract

HTTP clients often encounter interrupted data transfers as a result of canceled requests or dropped connections. Prior to interruption, part of a representation may have been exchanged. To complete the data transfer of the entire representation, it is often desirable to issue subsequent requests that transfer only the remainder of the representation. HTTP range requests support this concept of resumable downloads from server to client. This document describes a mechanism that supports resumable uploads from client to server using HTTP.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-httpbis-resumable-upload/.

Discussion of this document takes place on the HTTP Working Group mailing list (mailto:ietf-http-wg@w3.org), which is archived at https://lists.w3.org/Archives/Public/ietf-http-wg/. Working Group information can be found at https://httpwg.org/.

Source for this draft and an issue tracker can be found at https://github.com/httpwg/http-extensions/labels/resumable-upload.

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 5 September 2024.

Copyright Notice

Table of Contents

1. Introduction

HTTP clients often encounter interrupted data transfers as a result of canceled requests or dropped connections. Prior to interruption, part of a representation (see Section 3.2 of [HTTP]) might have been exchanged. To complete the data transfer of the entire representation, it is often desirable to issue subsequent requests that transfer only the remainder of the representation. HTTP range requests (see Section 14 of [HTTP]) support this concept of resumable downloads from server to client.

HTTP methods such as POST or PUT can be used by clients to request processing of representation data enclosed in the request message. The transfer of representation data from client to server is often referred to as an upload. Uploads are just as likely as downloads to suffer from the effects of data transfer interruption. Humans can play a role in upload interruptions through manual actions such as pausing an upload. Regardless of the cause of an interruption, servers may have received part of the representation before its occurrence and it is desirable if clients can complete the data transfer by sending only the remainder of the representation. The process of sending additional parts of a representation using subsequent HTTP requests from client to server is herein referred to as a resumable upload.

Connection interruptions are common and the absence of a standard mechanism for resumable uploads has lead to a proliferation of custom solutions. Some of those use HTTP, while others rely on other transfer mechanisms entirely. An HTTP-based standard solution is desirable for such a common class of problem.

This document defines an optional mechanism for HTTP that enables resumable uploads in a way that is backwards-compatible with conventional HTTP uploads. When an upload is interrupted, clients can send subsequent requests to query the server state and use this information to send the remaining data. Alternatively, they can cancel the upload entirely. Different from ranged downloads, this protocol does not support transferring different parts of the same representation in parallel.

2. Conventions and Definitions

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.

The terms Byte Sequence, Item, String, Token, Integer, and Boolean are imported from [STRUCTURED-FIELDS].

The terms client and server are from [HTTP].

3. Overview

Resumable uploads are supported in HTTP through use of a temporary resource, an *upload resource*, that is separate from the resource being uploaded to (hereafter, the *target resource*) and specific to that upload. By interacting with the upload resource, a client can retrieve the current offset of the upload (Section 5), append to the upload (Section 6), and cancel the upload (Section 7).

The remainder of this section uses an example of a file upload to illustrate different interactions with the upload resource. Note, however, that HTTP message exchanges use representation data (see Section 8.1 of [HTTP]), which means that resumable uploads can be used with many forms of content -- not just static files.

3.1. Example 1: Complete upload of file with known size

In this example, the client first attempts to upload a file with a known size in a single HTTP request to the target resource. An interruption occurs and the client then attempts to resume the upload using subsequent HTTP requests to the upload resource.

1) The client notifies the server that it wants to begin an upload (Section 4). The server reserves the required resources to accept the upload from the client, and the client begins transferring the entire file in the request content.

An informational response can be sent to the client, which signals the server's support of resumable upload as well as the upload resource URL via the Location header field (Section 10.2.2 of [HTTP]).

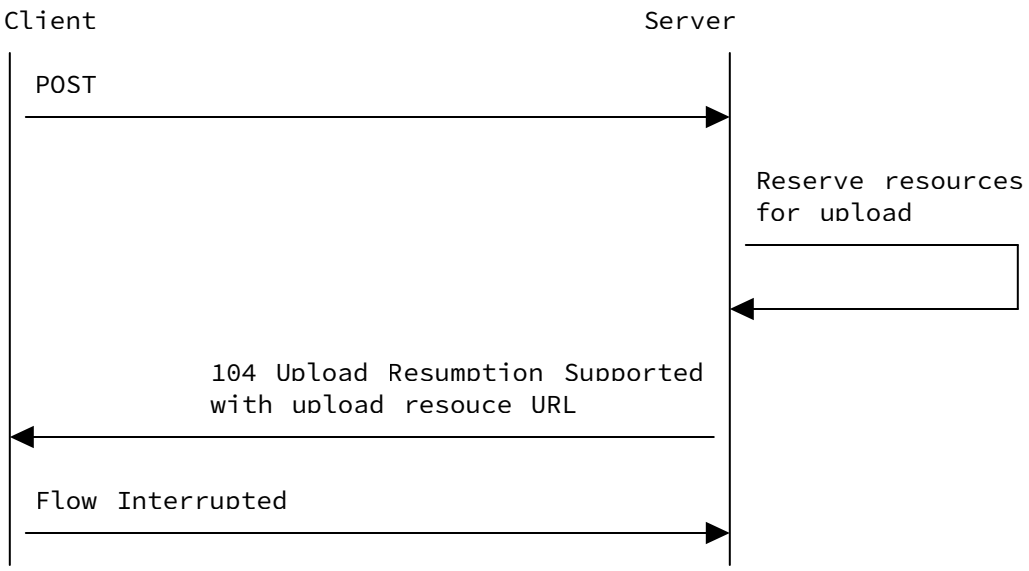

Figure 1: Upload Creation

2) If the connection to the server is interrupted, the client might want to resume the upload. However, before this is possible the client needs to know the amount of data that the server received before the interruption. It does so by retrieving the offset (Section 5) from the upload resource.

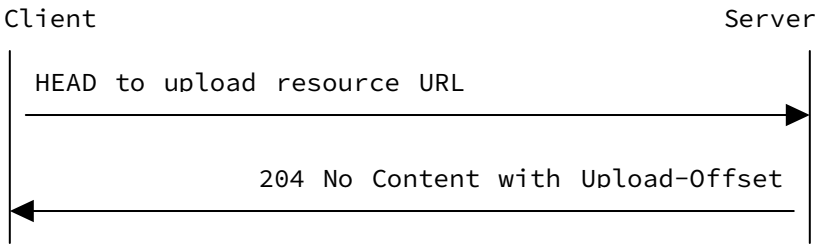

Figure 2: Offset Retrieval

3) The client can resume the upload by sending the remaining file content to the upload resource (Section 6), appending to the already stored data in the upload. The Upload-Offset value is included to ensure that the client and server agree on the offset that the upload resumes from.

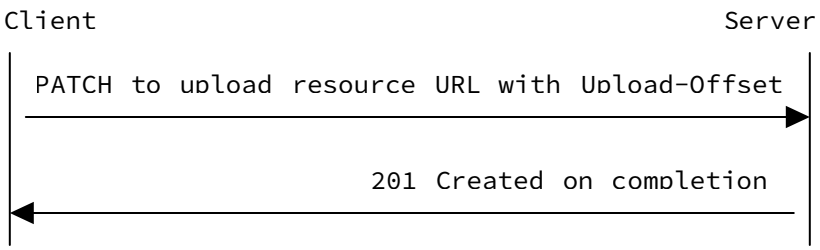

Figure 3: Upload Append

4) If the client is not interested in completing the upload, it can instruct the upload resource to delete the upload and free all related resources (Section 7).

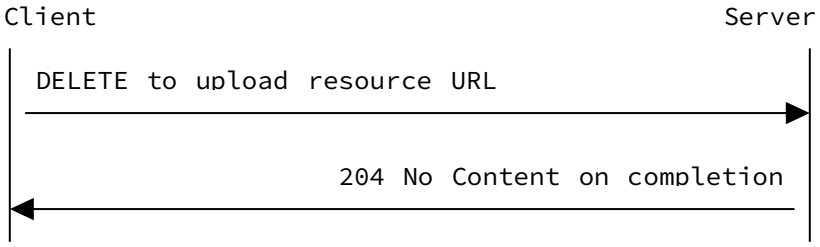

Figure 4: Upload Cancellation

3.2. Example 2: Upload as a series of parts

In some cases, clients might prefer to upload a file as a series of parts sent serially across multiple HTTP messages. One use case is to overcome server limits on HTTP message content size. Another use case is where the client does not know the final size, such as when file data originates from a streaming source.

This example shows how the client, with prior knowledge about the server's resumable upload support, can upload parts of a file incrementally.

1) If the client is aware that the server supports resumable upload, it can start an upload with the Upload-Complete field value set to false and the first part of the file.

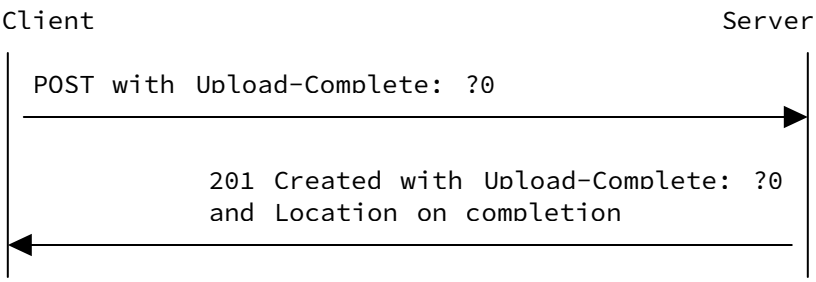

Figure 5: Incomplete Upload Creation

2) Subsequently, parts are appended (Section 6). The last part of the upload has a Upload-Complete field value set to true to indicate the complete transfer.

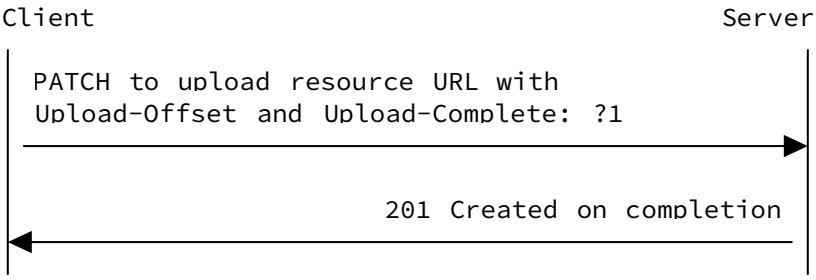

Figure 6: Upload Append Last Chunk

4. Upload Creation

When a resource supports resumable uploads, the first step is creating the upload resource. To be compatible with the widest range of resources, this is accomplished by including the Upload-Complete header field in the request that initiates the upload.

As a consequence, resumable uploads support all HTTP request methods that can carry content, such as POST, PUT, and PATCH. Similarly, the response to the upload request can have any status code. Both the method(s) and status code(s) supported are determined by the resource.

Upload-Complete **MUST** be set to false if the end of the request content is not the end of the upload. Otherwise, it **MUST** be set to true. This header field can be used for request identification by a server. The request **MUST NOT** include the Upload-Offset header field.

If the request is valid, the server **SHOULD** create an upload resource. Then, the server **MUST** include the Location header field in

the response and set its value to the URL of the upload resource. The client **MAY** use this URL for offset retrieval (Section 5), upload append (Section 6), and upload cancellation (Section 7).

Once the upload resource is available and while the request content is being uploaded, the target resource **MAY** send one or more informational responses with a 104 (Upload Resumption Supported) status code to the client. In the first informational response, the Location header field **MUST** be set to the URL pointing to the upload resource. In subsequent informational responses, the Location header field **MUST NOT** be set. An informational response **MAY** contain the Upload-Offset header field with the current upload offset as the value to inform the client about the upload progress. In subsequent informational responses, the upload offset **MUST NOT** be smaller than in previous informational responses. In addition, later offset retrievals (Section 5) **MUST NOT** receive an upload offset that is less than the offset reported in the latest informational response, allowing the client to free associated resources.

The server **MUST** send the Upload-Offset header field in the response if it considers the upload active, either when the response is a success (e.g. 201 (Created)), or when the response is a failure (e.g. 409 (Conflict)). The Upload-Offset field value **MUST** be equal to the end offset of the entire upload, or the begin offset of the next chunk if the upload is still incomplete. The client **SHOULD** consider the upload failed if the response has a status code that indicates a success but the offset indicated in the Upload-Offset field value does not equal the total of begin offset plus the number of bytes uploaded in the request.

If the request completes successfully and the entire upload is complete, the server **MUST** acknowledge it by responding with a successful status code between 200 and 299 (inclusive). Servers are **RECOMMENDED** to use 201 (Created) unless otherwise specified. The response **MUST NOT** include the Upload-Complete header field with the value of false.

If the request completes successfully but the entire upload is not yet complete, as indicated by an Upload-Complete field value of false in the request, the server **MUST** acknowledge it by responding with the 201 (Created) status code and an Upload-Complete header value set to false.

If the request includes an Upload-Complete field value set to true and a valid Content-Length header field, the client attempts to upload a fixed-length resource in one request. In this case, the upload's final size is the Content-Length field value and the server **MUST** record it to ensure its consistency.

The request content **MAY** be empty. If the Upload-Complete header field is then set to true, the client intends to upload an empty entity. An Upload-Complete header field is set to false is also valid. This can be used to create an upload resource URL before transferring data, which can save client or server resources. Since informational responses are optional, this technique provides another mechanism to learn the URL, at the cost of an additional round-trip before data upload can commence.

The following example shows an upload creation. The client transfers the entire 100 bytes in the first request. The server generates two informational responses to transmit the upload resource's URL and progress information, and one final response to acknowledge the completed upload:

```
POST /upload HTTP/1.1
Host: example.com
Upload-Draft-Interop-Version: 5
Upload-Complete: ?1
Content-Length: 100

[content (100 bytes)]

HTTP/1.1 104 Upload Resumption Supported
Upload-Draft-Interop-Version: 5
Location: https://example.com/upload/b530ce8ff

HTTP/1.1 104 Upload Resumption Supported
Upload-Draft-Interop-Version: 5
Upload-Offset: 50

HTTP/1.1 201 Created
Location: https://example.com/upload/b530ce8ff
Upload-Offset: 100
```

The next example shows an upload creation, where only the first 25 bytes are transferred. The server acknowledges the received data and that the upload is not complete yet:

```
POST /upload HTTP/1.1
Host: example.com
Upload-Draft-Interop-Version: 5
Upload-Complete: ?0
Content-Length: 25

[partial content (25 bytes)]
```

```
HTTP/1.1 201 Created
Location: https://example.com/upload/b530ce8ff
Upload-Complete: ?0
Upload-Offset: 25
```

If the client received an informational response with the upload URL in the Location field value, it **MAY** automatically attempt upload resumption when the connection is terminated unexpectedly, or if a 5xx status is received. The client **SHOULD NOT** automatically retry if it receives a 4xx status code.

File metadata can affect how servers might act on the uploaded file. Clients can send representation metadata (see Section 8.3 of [HTTP]) in the request that starts an upload. Servers **MAY** interpret this metadata or **MAY** ignore it. The Content-Type header field (Section 8.3 of [HTTP]) can be used to indicate the MIME type of the file. The Content-Disposition header field ([RFC6266]) can be used to transmit a filename; if included, the parameters **SHOULD** be either filename, filename* or boundary.

4.1. Feature Detection

If the client has no knowledge of whether the resource supports resumable uploads, a resumable request can be used with some additional constraints. In particular, the Upload-Complete field value (Section 8.2) **MUST NOT** be false if the server support is unclear. This allows the upload to function as if it is a regular upload.

Servers **SHOULD** use the 104 (Upload Resumption Supported) informational response to indicate their support for a resumable upload request.

Clients **MUST NOT** attempt to resume an upload unless they receive 104 (Upload Resumption Supported) informational response, or have other out-of-band methods to determine server support for resumable uploads.

4.2. Draft Version Identification

> **RFC Editor's Note:** Please remove this section and Upload-Draft-Interop-Version from all examples prior to publication of a final version of this document.

The current interop version is 5.

Client implementations of draft versions of the protocol **MUST** send a header field Upload-Draft-Interop-Version with the interop version as its value to its requests. The Upload-Draft-Interop-Version field value is an Integer.

Server implementations of draft versions of the protocol **MUST NOT** send a 104 (Upload Resumption Supported) informational response when the interop version indicated by the Upload-Draft-Interop-Version header field in the request is missing or mismatching.

Server implementations of draft versions of the protocol **MUST** also send a header field Upload-Draft-Interop-Version with the interop version as its value to the 104 (Upload Resumption Supported) informational response.

Client implementations of draft versions of the protocol **MUST** ignore a 104 (Upload Resumption Supported) informational response with missing or mismatching interop version indicated by the Upload-Draft-Interop-Version header field.

The reason both the client and the server are sending and checking the draft version is to ensure that implementations of the final RFC will not accidentally interop with draft implementations, as they will not check the existence of the Upload-Draft-Interop-Version header field.

5. Offset Retrieval

If an upload is interrupted, the client **MAY** attempt to fetch the offset of the incomplete upload by sending a HEAD request to the upload resource.

The request **MUST NOT** include an Upload-Offset or Upload-Complete header field. The server **MUST** reject requests with either of these fields by responding with a 400 (Bad Request) status code.

If the server considers the upload resource to be active, it **MUST** respond with a 204 (No Content) or 200 (OK) status code. The response **MUST** include the Upload-Offset header field, with the value set to the current resumption offset for the target resource. The response **MUST** include the Upload-Complete header field; the value is set to true only if the upload is complete.

An upload is considered complete only if the server completely and successfully received a corresponding creation request (Section 4) or append request (Section 6) with the Upload-Complete header value set to true.

The client **MUST NOT** perform offset retrieval while creation (Section 4) or append (Section 6) is in progress.

The offset **MUST** be accepted by a subsequent append (Section 6). Due to network delay and reordering, the server might still be receiving data from an ongoing transfer for the same upload resource, which in the client perspective has failed. The server **MAY** terminate any

transfers for the same upload resource before sending the response by abruptly terminating the HTTP connection or stream. Alternatively, the server **MAY** keep the ongoing transfer alive but ignore further bytes received past the offset.

The client **MUST NOT** start more than one append (Section 6) based on the resumption offset from a single offset retrieving (Section 5) request.

In order to prevent HTTP caching, the response **SHOULD** include a Cache-Control header field with the value no-store.

If the server does not consider the upload resource to be active, it **MUST** respond with a 404 (Not Found) status code.

The resumption offset can be less than or equal to the number of bytes the client has already sent. The client **MAY** reject an offset which is greater than the number of bytes it has already sent during this upload. The client is expected to handle backtracking of a reasonable length. If the offset is invalid for this upload, or if the client cannot backtrack to the offset and reproduce the same content it has already sent, the upload **MUST** be considered a failure. The client **MAY** cancel the upload (Section 7) after rejecting the offset.

The following example shows an offset retrieval request. The server indicates the new offset and that the upload is not complete yet:

```
HEAD /upload/b530ce8ff HTTP/1.1
Host: example.com
Upload-Draft-Interop-Version: 5

HTTP/1.1 204 No Content
Upload-Offset: 100
Upload-Complete: ?0
Cache-Control: no-store
```

The client **SHOULD NOT** automatically retry if a client error status code between 400 and 499 (inclusive) is received.

6. Upload Append

Upload appending is used for resuming an existing upload.

The request **MUST** use the PATCH method and be sent to the upload resource. The Upload-Offset field value (Section 8.1) **MUST** be set to the resumption offset.

If the end of the request content is not the end of the upload, the Upload-Complete field value (Section 8.2) **MUST** be set to false.

The server **SHOULD** respect representation metadata received during creation (Section 4) and ignore any representation metadata received from appending (Section 6).

If the server does not consider the upload associated with the upload resource active, it **MUST** respond with a 404 (Not Found) status code.

The client **MUST NOT** perform multiple upload transfers for the same upload resource in parallel. This helps avoid race conditions, and data loss or corruption. The server is **RECOMMENDED** to take measures to avoid parallel upload transfers: The server **MAY** terminate any creation (Section 4) or append (Section 6) for the same upload URL. Since the client is not allowed to perform multiple transfers in parallel, the server can assume that the previous attempt has already failed. Therefore, the server **MAY** abruptly terminate the previous HTTP connection or stream.

If the offset indicated by the Upload-Offset field value does not match the offset provided by the immediate previous offset retrieval (Section 5), or the end offset of the immediate previous incomplete successful transfer, the server **MUST** respond with a 409 (Conflict) status code.

While the request content is being uploaded, the target resource **MAY** send one or more informational responses with a 104 (Upload Resumption Supported) status code to the client. These informational responses **MUST NOT** contain the Location header field. They **MAY** include the Upload-Offset header field with the current upload offset as the value to inform the client about the upload progress. The same restrictions on the Upload-Offset header field in informational responses from the upload creation (Section 4) apply.

The server **MUST** send the Upload-Offset header field in the response if it considers the upload active, either when the response is a success (e.g. 201 (Created)), or when the response is a failure (e.g. 409 (Conflict)). The value **MUST** be equal to the end offset of the entire upload, or the begin offset of the next chunk if the upload is still incomplete. The client **SHOULD** consider the upload failed if the status code indicates a success but the offset indicated by the Upload-Offset field value does not equal the total of begin offset plus the number of bytes uploaded in the request.

If the request completes successfully and the entire upload is complete, the server **MUST** acknowledge it by responding with a successful status code between 200 and 299 (inclusive). Servers are **RECOMMENDED** to use a 201 (Created) response if not otherwise specified. The response **MUST NOT** include the Upload-Complete header field with the value set to false.

If the request completes successfully but the entire upload is not yet complete indicated by the Upload-Complete field value set to false, the server **MUST** acknowledge it by responding with a 201 (Created) status code and the Upload-Complete field value set to true.

If the request includes the Upload-Complete field value set to true and a valid Content-Length header field, the client attempts to upload the remaining resource in one request. In this case, the upload's final size is the sum of the upload's offset and the Content-Length header field. If the server does not have a record of the upload's final size from creation or the previous append, the server **MUST** record the upload's final size to ensure its consistency. If the server does have a previous record, that value **MUST** match the upload's final size. If they do not match, the server **MUST** reject the request with a 400 (Bad Request) status code.

The request content **MAY** be empty. If the Upload-Complete field is then set to true, the client wants to complete the upload without appending additional data.

The following example shows an upload append. The client transfers the next 100 bytes at an offset of 100 and does not indicate that the upload is then completed. The server acknowledges the new offset:

```
PATCH /upload/b530ce8ff HTTP/1.1
Host: example.com
Upload-Offset: 100
Upload-Draft-Interop-Version: 5
Content-Length: 100

[content (100 bytes)]

HTTP/1.1 201 Created
Upload-Offset: 200
```

The client **MAY** automatically attempt upload resumption when the connection is terminated unexpectedly, or if a server error status code between 500 and 599 (inclusive) is received. The client **SHOULD NOT** automatically retry if a client error status code between 400 and 499 (inclusive) is received.

7. Upload Cancellation

If the client wants to terminate the transfer without the ability to resume, it can send a DELETE request to the upload resource. Doing so is an indication that the client is no longer interested in continuing the upload, and that the server can release any resources associated with it.

The client **MUST NOT** initiate cancellation without the knowledge of server support.

The request **MUST** use the DELETE method. The request **MUST NOT** include an Upload-Offset or Upload-Complete header field. The server **MUST** reject the request with a Upload-Offset or Upload-Complete header field with a 400 (Bad Request) status code.

If the server successfully deactivates the upload resource, it **MUST** respond with a 204 (No Content) status code.

The server **MAY** terminate any in-flight requests to the upload resource before sending the response by abruptly terminating their HTTP connection(s) or stream(s).

If the server does not consider the upload resource to be active, it **MUST** respond with a 404 (Not Found) status code.

If the server does not support cancellation, it **MUST** respond with a 405 (Method Not Allowed) status code.

The following example shows an upload cancellation:

```
DELETE /upload/b530ce8ff HTTP/1.1
Host: example.com
Upload-Draft-Interop-Version: 5

HTTP/1.1 204 No Content
```

8. Header Fields

8.1. Upload-Offset

The Upload-Offset request and response header field indicates the resumption offset of corresponding upload, counted in bytes. The Upload-Offset field value is an Integer.

8.2. Upload-Complete

The Upload-Complete request and response header field indicates whether the corresponding upload is considered complete. The Upload-Complete field value is a Boolean.

The Upload-Complete header field **MUST** only be used if support by the resource is known to the client (Section 4.1).

9. Redirection

The 301 (Moved Permanently) and 302 (Found) status codes **MUST NOT** be used in offset retrieval (Section 5) and upload cancellation

(Section 7) responses. For other responses, the upload resource **MAY** return a 308 (Permanent Redirect) status code and clients **SHOULD** use new permanent URI for subsequent requests. If the client receives a 307 (Temporary Redirect) response to an offset retrieval (Section 5) request, it **MAY** apply the redirection directly in an immediate subsequent upload append (Section 6).

10. Security Considerations

The upload resource URL is the identifier used for modifying the upload. Without further protection of this URL, an attacker may obtain information about an upload, append data to it, or cancel it. To prevent this, the server **SHOULD** ensure that only authorized clients can access the upload resource. In addition, the upload resource URL **SHOULD** be generated in such a way that makes it hard to be guessed by unauthorized clients.

Some servers or intermediaries provide scanning of content uploaded by clients. Any scanning mechanism that relies on receiving a complete file in a single request message can be defeated by resumable uploads because content can be split across multiple messages. Servers or intermediaries wishing to perform content scanning **SHOULD** consider how resumable uploads can circumvent scanning and take appropriate measures. Possible strategies include waiting for the upload to complete before scanning a full file, or disabling resumable uploads.

Resumable uploads are vulnerable to Slowloris-style attacks [SLOWLORIS]. A malicious client may create upload resources and keep them alive by regularly sending PATCH requests with no or small content to the upload resources. This could be abused to exhaust server resources by creating and holding open uploads indefinitely with minimal work.

Servers **SHOULD** provide mitigations for Slowloris attacks, such as increasing the maximum number of clients the server will allow, limiting the number of uploads a single client is allowed to make, imposing restrictions on the minimum transfer speed an upload is allowed to have, and restricting the length of time an upload resource can exist.

11. IANA Considerations

IANA is asked to register the following entries in the "Hypertext Transfer Protocol (HTTP) Field Name Registry":

Field Name	Status	Reference
Upload-Complete	permanent	Section 8.2 of this document
Upload-Offset	permanent	Section 8.1 of this document

Table 1

IANA is asked to register the following entry in the "HTTP Status Codes" registry:

Value: 104 (suggested value)

Description: Upload Resumption Supported

Specification: This document

12. References

12.1. Normative References

[HTTP] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/rfc/rfc9110>.

[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/rfc/rfc2119>.

[RFC6266] Reschke, J., "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)", RFC 6266, DOI 10.17487/RFC6266, June 2011, <https://www.rfc-editor.org/rfc/rfc6266>.

[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/rfc/rfc8174>.

[STRUCTURED-FIELDS] Nottingham, M. and P. Kamp, "Structured Field Values for HTTP", RFC 8941, DOI 10.17487/RFC8941, February 2021, <https://www.rfc-editor.org/rfc/rfc8941>.

12.2. Informative References

[SLOWLORIS] "RSnake" Hansen, R., "Welcome to Slowloris - the low bandwidth, yet greedy and poisonous HTTP client!", June 2009, <https://web.archive.org/web/20150315054838/http://ha.ckers.org/slowloris/>.

Appendix A. Informational Response

The server is allowed to respond to upload creation (Section 4) requests with a 104 (Upload Resumption Supported) intermediate response as soon as the server has validated the request. This way,

the client knows that the server supports resumable uploads before the complete response is received. The benefit is the clients can defer starting the actual data transfer until the server indicates full support (i.e. resumable are supported, the provided upload URL is active etc).

On the contrary, support for intermediate responses (the 1XX range) in existing software is limited or not at all present. Such software includes proxies, firewalls, browsers, and HTTP libraries for clients and server. Therefore, the 104 (Upload Resumption Supported) status code is optional and not mandatory for the successful completion of an upload. Otherwise, it might be impossible in some cases to implement resumable upload servers using existing software packages. Furthermore, as parts of the current internet infrastructure currently have limited support for intermediate responses, a successful delivery of a 104 (Upload Resumption Supported) from the server to the client should be assumed.

We hope that support for intermediate responses increases in the near future, to allow a wider usage of 104 (Upload Resumption Supported).

Appendix B. Feature Detection

This specification includes a section about feature detection (it was called service discovery in earlier discussions, but this name is probably ill-suited). The idea is to allow resumable uploads to be transparently implemented by HTTP clients. This means that application developers just keep using the same API of their HTTP library as they have done in the past with traditional, non-resumable uploads. Once the HTTP library gets updated (e.g. because mobile OS or browsers start implementing resumable uploads), the HTTP library can transparently decide to use resumable uploads without explicit configuration by the application developer. Of course, in order to use resumable uploads, the HTTP library needs to know whether the server supports resumable uploads. If no support is detected, the HTTP library should use the traditional, non-resumable upload technique. We call this process feature detection.

Ideally, the technique used for feature detection meets following **criteria** (there might not be one approach which fits all requirements, so we have to prioritize them):

1. Avoid additional roundtrips by the client, if possible (i.e. an additional HTTP request by the client should be avoided).

2. Be backwards compatible to HTTP/1.1 and existing network infrastructure: This means to avoid using new features in HTTP/

2, or features which might require changes to existing network infrastructure (e.g. nginx or HTTP libraries)

3. Conserve the user's privacy (i.e. the feature detection should not leak information to other third-parties about which URLs have been connected to)

Following **approaches** have already been considered in the past. All except the last approaches have not been deemed acceptable and are therefore not included in the specification. This follow list is a reference for the advantages and disadvantages of some approaches:

Include a support statement in the SETTINGS frame. The SETTINGS frame is a HTTP/2 feature and is sent by the server to the client to exchange information about the current connection. The idea was to include an additional statement in this frame, so the client can detect support for resumable uploads without an additional roundtrip. The problem is that this is not compatible with HTTP/1.1. Furthermore, the SETTINGS frame is intended for information about the current connection (not bound to a request/response) and might not be persisted when transmitted through a proxy.

Include a support statement in the DNS record. The client can detect support when resolving a domain name. Of course, DNS is not semantically the correct layer. Also, DNS might not be involved if the record is cached or retrieved from a hosts files.

Send a HTTP request to ask for support. This is the easiest approach where the client sends an OPTIONS request and uses the response to determine if the server indicates support for resumable uploads. An alternative is that the client sends the request to a well-known URL to obtain this response, e.g. /.well-known/resumable-uploads. Of course, while being fully backwards-compatible, it requires an additional roundtrip.

Include a support statement in previous responses. In many cases, the file upload is not the first time that the client connects to the server. Often additional requests are sent beforehand for authentication, data retrieval etc. The responses for those requests can also include a header field which indicates support for resumable uploads. There are two options: - Use the standardized Alt-Svc response header field. However, it has been indicated to us that this header field might be reworked in the future and could also be semantically different from our intended usage. - Use a new response header field Resumable-Uploads: https://example.org/files/* to indicate under which endpoints support for resumable uploads is available.

Send a 104 intermediate response to indicate support. The clients normally starts a traditional upload and includes a header field indicate that it supports resumable uploads (e.g. Upload-Offset: 0). If the server also supports resumable uploads, it will immediately respond with a 104 intermediate response to indicate its support, before further processing the request. This way the client is informed during the upload whether it can resume from possible connection errors or not. While an additional roundtrip is avoided, the problem with that solution is that many HTTP server libraries do not support sending custom 1XX responses and that some proxies may not be able to handle new 1XX status codes correctly.

Send a 103 Early Hint response to indicate support. This approach is the similar to the above one, with one exception: Instead of a new 104 (Upload Resumption Supported) status code, the existing 103 (Early Hint) status code is used in the intermediate response. The 103 code would then be accompanied by a header field indicating support for resumable uploads (e.g. Resumable-Uploads: 1). It is unclear whether the Early Hints code is appropriate for that, as it is currently only used to indicate resources for prefetching them.

Appendix C. Upload Metadata

When an upload is created (Section 4), the Content-Type and Content-Disposition header fields are allowed to be included. They are intended to be a standardized way of communicating the file name and file type, if available. However, this is not without controversy. Some argue that since these header fields are already defined in other specifications, it is not necessary to include them here again. Furthermore, the Content-Disposition header field's format is not clearly enough defined. For example, it is left open which disposition value should be used in the header field. There needs to be more discussion whether this approach is suited or not.

However, from experience with the tus project, users are often asking for a way to communicate the file name and file type. Therefore, we believe it is help to explicitly include an approach for doing so.

Appendix D. FAQ

* **Are multipart requests supported?** Yes, requests whose content is encoded using the multipart/form-data are implicitly supported. The entire encoded content can be considered as a single file, which is then uploaded using the resumable protocol. The server, of course, must store the delimiter ("boundary") separating each part and must be able to parse the multipart format once the upload is completed.

Acknowledgments

This document is based on an Internet-Draft specification written by Jiten Mehta, Stefan Matsson, and the authors of this document.

The tus v1 protocol is a specification for a resumable file upload protocol over HTTP. It inspired the early design of this protocol. Members of the tus community helped significantly in the process of bringing this work to the IETF.

The authors would like to thank Mark Nottingham for substantive contributions to the text.

Changes

This section is to be removed before publishing as an RFC.

Since draft-ietf-httpbis-resumable-upload-02

- Add upload progress notifications via informational responses.
- Add security consideration regarding request filtering.
- Explain the use of empty requests for creation uploads and appending.
- Extend security consideration to include resource exhaustion attacks.
- Allow 200 status codes for offset retrieval.
- Increase the draft interop version.

Since draft-ietf-httpbis-resumable-upload-01

- Replace Upload-Incomplete header with Upload-Complete.
- Replace terminology about procedures with HTTP resources.
- Increase the draft interop version.

Since draft-ietf-httpbis-resumable-upload-00

- Remove Upload-Token and instead use Server-generated upload URL for upload identification.
- Require the Upload-Incomplete header field in Upload Creation Procedure.
- Increase the draft interop version.

Since draft-tus-httpbis-resumable-uploads-protocol-02

None

Since draft-tus-httpbis-resumable-uploads-protocol-01

* Clarifying backtracking and preventing skipping ahead during the Offset Receiving Procedure.

* Clients auto-retry 404 is no longer allowed.

Since draft-tus-httpbis-resumable-uploads-protocol-00

* Split the Upload Transfer Procedure into the Upload Creation Procedure and the Upload Appending Procedure.

Authors' Addresses

Marius Kleidl (editor)
Transloadit

Email: marius@transloadit.com

Guoye Zhang (editor)
Apple Inc.

Email: guoye_zhang@apple.com

Lucas Pardue (editor)
Cloudflare

Email: lucas@lucaspardue.com