HTTPbis Working Group	R. Fielding, Ed.
Internet-Draft	Adobe
Updates: 2817 (if approved)	J. Gettys
Obsoletes: 2616 (if approved)	Alcatel-Lucent
Intended status: Standards Track	J. Mogul
Expires: October 20, 2011	HP
	H. Frystyk
	Microsoft
	L. Masinter
	Adobe
	P. Leach
	Microsoft
	T. Berners-Lee
	W3C/MIT
	Y. Lafon, Ed.
	W3C
	J. F. Reschke, Ed.
	greenbytes
	April 18, 2011

HTTP/1.1, part 2: Message Semantics
draft-ietf-httpbis-p2-semantics-14

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 2 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 2 defines the semantics of HTTP messages as expressed by request methods, request header fields, response status codes, and response header fields.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft should take place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org), which is archived at http://lists.w3.org/Archives/Public/ietf-http-wg/.
The current issues list is at http://tools.ietf.org/wg/httpbis/trac/report/3 and related documents (including fancy diffs) can be found at http://tools.ietf.org/wg/httpbis/.
The changes in this draft are summarized in Appendix Appendix C.15.

Status of this Memo

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

Copyright Notice

Table of Contents

1. Introduction

This document defines HTTP/1.1 request and response semantics. Each HTTP message, as defined in [Part1], is in the form of either a request or a response. An HTTP server listens on a connection for HTTP requests and responds to each request, in the order received on that connection, with one or more HTTP response messages. This document defines the commonly agreed upon semantics of the HTTP uniform interface, the intentions defined by each request method, and the various response messages that might be expected as a result of applying that method to the target resource.
This document is currently disorganized in order to minimize the changes between drafts and enable reviewers to see the smaller errata changes. A future draft will reorganize the sections to better reflect the content. In particular, the sections will be ordered according to the typical processing of an HTTP request message (after message parsing): resource mapping, methods, request modifying header fields, response status, status modifying header fields, and resource metadata. The current mess reflects how widely dispersed these topics and associated requirements had become in [RFC2616].

1.1. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].
An implementation is not compliant if it fails to satisfy one or more of the "MUST" or "REQUIRED" level requirements for the protocols it implements. An implementation that satisfies all the "MUST" or "REQUIRED" level and all the "SHOULD" level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies

all the "MUST" level requirements but not all the "SHOULD" level requirements for its protocols is said to be "conditionally compliant".

1.2. Syntax Notation

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). Appendix Appendix B shows the collected ABNF, with the list rule expanded.
The following core rules are included by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), VCHAR (any visible USASCII character), and WSP (whitespace).

1.2.1. Core Rules

The core rules below are defined in Section 1.2.2 of [Part1]:

```
  quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
  token         = <token, defined in [Part1], Section 1.2.2>
  OWS           = <OWS, defined in [Part1], Section 1.2.2>
  RWS           = <RWS, defined in [Part1], Section 1.2.2>
  obs-text      = <obs-text, defined in [Part1], Section 1.2.2>
```

1.2.2. ABNF Rules defined in other Parts of the Specification

The ABNF rules below are defined in other parts:

```
  absolute-URI  = <absolute-URI, defined in [Part1], Section 2.6>
  comment       = <comment, defined in [Part1], Section 3.2>
  HTTP-date     = <HTTP-date, defined in [Part1], Section 6.1>
  partial-URI   = <partial-URI, defined in [Part1], Section 2.6>
  product       = <product, defined in [Part1], Section 6.3>
  URI-reference = <URI-reference, defined in [Part1], Section 2.6>
```

2. Method

The Method token indicates the request method to be performed on the target resource (Section 4.3 of [Part1]). The method is case-sensitive.

```
  Method         = token
```

The list of methods allowed by a resource can be specified in an Allow header field (Section 9.1). The status code of the response always notifies the client whether a method is currently allowed on a resource, since the set of allowed methods can change dynamically. An origin server SHOULD respond with the status code 405 (Method Not

Allowed) if the method is known by the origin server but not allowed for the resource, and 501 (Not Implemented) if the method is unrecognized or not implemented by the origin server. The methods GET and HEAD MUST be supported by all general-purpose servers. All other methods are OPTIONAL; however, if the above methods are implemented, they MUST be implemented with the same semantics as those specified in Section 7.

2.1. Overview of Methods

The methods listed below are defined in Section 7.

Method Name	Defined in...
OPTIONS	Section 7.2
GET	Section 7.3
HEAD	Section 7.4
POST	Section 7.5
PUT	Section 7.6
DELETE	Section 7.7
TRACE	Section 7.8
CONNECT	Section 7.9

Note that this list is not exhaustive — it does not include request methods defined in other specifications.

2.2. Method Registry

The HTTP Method Registry defines the name space for the Method token in the Request line of an HTTP request.
Registrations MUST include the following fields:

*Method Name (see Section 2)

*Safe ("yes" or "no", see Section 7.1.1)

*Pointer to specification text

Values to be added to this name space are subject to IETF review ([RFC5226], Section 4.1).
The registry itself is maintained at http://www.iana.org/assignments/http-methods.

2.2.1. Considerations for New Methods

When it is necessary to express new semantics for a HTTP request that aren't specific to a single application or media type, and currently

defined methods are inadequate, it may be appropriate to register a new method.
HTTP methods are generic; that is, they are potentially applicable to any resource, not just one particular media type, "type" of resource, or application. As such, it is preferred that new HTTP methods be registered in a document that isn't specific to a single application, so that this is clear.
Due to the parsing rules defined in Section 3.3 of [Part1], definitions of HTTP methods cannot prohibit the presence of a message-body on either the request or the response message (with responses to HEAD requests being the single exception). Definitions of new methods cannot change this rule, but they can specify that only zero-length bodies (as opposed to absent bodies) are allowed.
New method definitions need to indicate whether they are safe (Section 7.1.1), what semantics (if any) the request body has, and whether they are idempotent (Section 7.1.2). They also need to state whether they can be cached ([Part6]); in particular what conditions a cache may store the response, and under what conditions such a stored response may be used to satisfy a subsequent request.

3. Request Header Fields

The request header fields allow the client to pass additional information about the request, and about the client itself, to the server. These fields act as request modifiers, with semantics equivalent to the parameters on a programming language method invocation.

Header Field Name	Defined in...
Accept	Section 6.1 of [Part3]
Accept-Charset	Section 6.2 of [Part3]
Accept-Encoding	Section 6.3 of [Part3]
Accept-Language	Section 6.4 of [Part3]
Authorization	Section 4.1 of [Part7]
Expect	Section 9.2
From	Section 9.3
Host	Section 9.4 of [Part1]
If-Match	Section 3.1 of [Part4]
If-Modified-Since	Section 3.3 of [Part4]
If-None-Match	Section 3.2 of [Part4]
If-Range	Section 5.3 of [Part5]
If-Unmodified-Since	Section 3.4 of [Part4]
Max-Forwards	Section 9.5

Header Field Name	Defined in...
Proxy-Authorization	Section 4.3 of [Part7]
Range	Section 5.4 of [Part5]
Referer	Section 9.6
TE	Section 9.5 of [Part1]
User-Agent	Section 9.9

4. Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request.
The Reason-Phrase is intended to give a short textual description of the Status-Code and is intended for a human user. The client does not need to examine or display the Reason-Phrase.

```
  Status-Code    = 3DIGIT
  Reason-Phrase  = *( WSP / VCHAR / obs-text )
```

HTTP status codes are extensible. HTTP applications are not required to understand the meaning of all registered status codes, though such understanding is obviously desirable. However, applications MUST understand the class of any status code, as indicated by the first digit, and treat any unrecognized response as being equivalent to the x00 status code of that class, with the exception that an unrecognized response MUST NOT be cached. For example, if an unrecognized status code of 431 is received by the client, it can safely assume that there was something wrong with its request and treat the response as if it had received a 400 status code. In such cases, user agents SHOULD present to the user the representation enclosed with the response, since that representation is likely to include human-readable information which will explain the unusual status.

4.1. Overview of Status Codes

The status codes listed below are defined in Section 8 of this specification, Section 4 of [Part4], Section 3 of [Part5], and Section 3 of [Part7]. The reason phrases listed here are only recommendations — they can be replaced by local equivalents without affecting the protocol.

Status-Code	Reason-Phrase	Defined in...
100	Continue	Section 8.1.1
101	Switching Protocols	Section 8.1.2
200	OK	Section 8.2.1
201	Created	Section 8.2.2

Status-Code	Reason-Phrase	Defined in...
202	Accepted	Section 8.2.3
203	Non-Authoritative Information	Section 8.2.4
204	No Content	Section 8.2.5
205	Reset Content	Section 8.2.6
206	Partial Content	Section 3.1 of [Part5]
300	Multiple Choices	Section 8.3.1
301	Moved Permanently	Section 8.3.2
302	Found	Section 8.3.3
303	See Other	Section 8.3.4
304	Not Modified	Section 4.1 of [Part4]
305	Use Proxy	Section 8.3.6
307	Temporary Redirect	Section 8.3.8
400	Bad Request	Section 8.4.1
401	Unauthorized	Section 3.1 of [Part7]
402	Payment Required	Section 8.4.3
403	Forbidden	Section 8.4.4
404	Not Found	Section 8.4.5
405	Method Not Allowed	Section 8.4.6
406	Not Acceptable	Section 8.4.7
407	Proxy Authentication Required	Section 3.2 of [Part7]
408	Request Time-out	Section 8.4.9
409	Conflict	Section 8.4.10
410	Gone	Section 8.4.11
411	Length Required	Section 8.4.12
412	Precondition Failed	Section 4.2 of [Part4]
413	Request Entity Too Large	Section 8.4.14
414	URI Too Long	Section 8.4.15
415	Unsupported Media Type	Section 8.4.16
416	Requested range not satisfiable	Section 3.2 of [Part5]
417	Expectation Failed	Section 8.4.18
426	Upgrade Required	Section 8.4.19
500	Internal Server Error	Section 8.5.1
501	Not Implemented	Section 8.5.2
502	Bad Gateway	Section 8.5.3

Status-Code	Reason-Phrase	Defined in...
503	Service Unavailable	Section 8.5.4
504	Gateway Time-out	Section 8.5.5
505	HTTP Version not supported	Section 8.5.6

Note that this list is not exhaustive — it does not include extension status codes defined in other specifications.

4.2. Status Code Registry

The HTTP Status Code Registry defines the name space for the Status-Code token in the Status-Line of an HTTP response.
Values to be added to this name space are subject to IETF review ([RFC5226], Section 4.1).
The registry itself is maintained at http://www.iana.org/assignments/http-status-codes.

4.2.1. Considerations for New Status Codes

When it is necessary to express new semantics for a HTTP response that aren't specific to a single application or media type, and currently defined status codes are inadequate, a new status code can be registered.
HTTP status codes are generic; that is, they are potentially applicable to any resource, not just one particular media type, "type" of resource, or application. As such, it is preferred that new HTTP status codes be registered in a document that isn't specific to a single application, so that this is clear.
Definitions of new HTTP status codes typically explain the request conditions that produce a response containing the status code (e.g., combinations of request headers and/or method(s)), along with any interactions with response headers (e.g., those that are required, those that modify the semantics of the response).
New HTTP status codes are required to fall under one of the categories defined in Section 8. To allow existing parsers to properly handle them, new status codes cannot disallow a response body, although they can mandate a zero-length response body. They can require the presence of one or more particular HTTP response header(s).
Likewise, their definitions can specify that caches are allowed to use heuristics to determine their freshness (see [Part6]; by default, it is not allowed), and can define how to determine the resource which they carry a representation for (see Section 6.1; by default, it is anonymous).

5. Response Header Fields

The response header fields allow the server to pass additional information about the response which cannot be placed in the Status-

Line. These header fields give information about the server and about further access to the target resource (Section 4.3 of [Part1]).

Header Field Name	Defined in...
Accept-Ranges	Section 5.1 of [Part5]
Age	Section 3.1 of [Part6]
Allow	Section 9.1
ETag	Section 2.2 of [Part4]
Location	Section 9.4
Proxy-Authenticate	Section 4.2 of [Part7]
Retry-After	Section 9.7
Server	Section 9.8
Vary	Section 3.5 of [Part6]
WWW-Authenticate	Section 4.4 of [Part7]

6. Representation

Request and Response messages MAY transfer a representation if not otherwise restricted by the request method or response status code. A representation consists of metadata (representation header fields) and data (representation body). When a complete or partial representation is enclosed in an HTTP message, it is referred to as the payload of the message. HTTP representations are defined in [Part3].
A representation body is only present in a message when a message-body is present, as described in Section 3.3 of [Part1]. The representation body is obtained from the message-body by decoding any Transfer-Encoding that might have been applied to ensure safe and proper transfer of the message.

6.1. Identifying the Resource Associated with a Representation

It is sometimes necessary to determine an identifier for the resource associated with a representation.
An HTTP request representation, when present, is always associated with an anonymous (i.e., unidentified) resource.
In the common case, an HTTP response is a representation of the target resource (see Section 4.3 of [Part1]). However, this is not always the case. To determine the URI of the resource a response is associated with, the following rules are used (with the first applicable one being selected):

1. If the response status code is 200 or 203 and the request method was GET, the response payload is a representation of the target resource.

2. If the response status code is 204, 206, or 304 and the request method was GET or HEAD, the response payload is a partial representation of the target resource (see Section 2.8 of [Part6]).

3. If the response has a Content-Location header field, and that URI is the same as the effective request URI, the response payload is a representation of the target resource.

4. If the response has a Content-Location header field, and that URI is not the same as the effective request URI, then the response asserts that its payload is a representation of the resource identified by the Content-Location URI. However, such an assertion cannot be trusted unless it can be verified by other means (not defined by HTTP).

5. Otherwise, the response is a representation of an anonymous (i.e., unidentified) resource.

7. Method Definitions

The set of common request methods for HTTP/1.1 is defined below. Although this set can be expanded, additional methods cannot be assumed to share the same semantics for separately extended clients and servers.

7.1. Safe and Idempotent Methods

7.1.1. Safe Methods

Implementors need to be aware that the software represents the user in their interactions over the Internet, and need to allow the user to be aware of any actions they take which might have an unexpected significance to themselves or others.
In particular, the convention has been established that the GET, HEAD, OPTIONS, and TRACE request methods SHOULD NOT have the significance of taking an action other than retrieval. These request methods ought to be considered "safe". This allows user agents to represent other methods, such as POST, PUT and DELETE, in a special way, so that the user is made aware of the fact that a possibly unsafe action is being requested.
Naturally, it is not possible to ensure that the server does not generate side-effects as a result of performing a GET request; in fact, some dynamic resources consider that a feature. The important distinction here is that the user did not request the side-effects, so therefore cannot be held accountable for them.

7.1.2. Idempotent Methods

Request methods can also have the property of "idempotence" in that, aside from error or expiration issues, the intended effect of multiple identical requests is the same as for a single request. PUT, DELETE, and all safe request methods are idempotent. It is important to note that idempotence refers only to changes requested by the client: a server is free to change its state due to multiple requests for the purpose of tracking those requests, versioning of results, etc.

7.2. OPTIONS

The OPTIONS method requests information about the communication options available on the request/response chain identified by the effective request URI. This method allows a client to determine the options and/or requirements associated with a resource, or the capabilities of a server, without implying a resource action or initiating a resource retrieval.

Responses to the OPTIONS method are not cacheable.

If the OPTIONS request includes a message-body (as indicated by the presence of Content-Length or Transfer-Encoding), then the media type MUST be indicated by a Content-Type field. Although this specification does not define any use for such a body, future extensions to HTTP might use the OPTIONS body to make more detailed queries on the server.

If the request-target is an asterisk ("*"), the OPTIONS request is intended to apply to the server in general rather than to a specific resource. Since a server's communication options typically depend on the resource, the "*" request is only useful as a "ping" or "no-op" type of method; it does nothing beyond allowing the client to test the capabilities of the server. For example, this can be used to test a proxy for HTTP/1.1 compliance (or lack thereof).

If the request-target is not an asterisk, the OPTIONS request applies only to the options that are available when communicating with that resource.

A 200 response SHOULD include any header fields that indicate optional features implemented by the server and applicable to that resource (e.g., Allow), possibly including extensions not defined by this specification. The response body, if any, SHOULD also include information about the communication options. The format for such a body is not defined by this specification, but might be defined by future extensions to HTTP. Content negotiation MAY be used to select the appropriate response format. If no response body is included, the response MUST include a Content-Length field with a field-value of "0".

The Max-Forwards header field MAY be used to target a specific proxy in the request chain (see Section 9.5). If no Max-Forwards field is present in the request, then the forwarded request MUST NOT include a Max-Forwards field.

7.3. GET

The GET method requests transfer of a current representation of the target resource.
If the target resource is a data-producing process, it is the produced data which shall be returned as the representation in the response and not the source text of the process, unless that text happens to be the output of the process.
The semantics of the GET method change to a "conditional GET" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. A conditional GET requests that the representation be transferred only under the circumstances described by the conditional header field(s). The conditional GET request is intended to reduce unnecessary network usage by allowing cached representations to be refreshed without requiring multiple requests or transferring data already held by the client.
The semantics of the GET method change to a "partial GET" if the request message includes a Range header field. A partial GET requests that only part of the representation be transferred, as described in Section 5.4 of [Part5]. The partial GET request is intended to reduce unnecessary network usage by allowing partially-retrieved representations to be completed without transferring data already held by the client.
Bodies on GET requests have no defined semantics. Note that sending a body on a GET request might cause some existing implementations to reject the request.
The response to a GET request is cacheable and MAY be used to satisfy subsequent GET and HEAD requests (see [Part6]).
See Section 11.2 for security considerations when used for forms.

7.4. HEAD

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metadata contained in the HTTP header fields in response to a HEAD request SHOULD be identical to the information sent in response to a GET request. This method can be used for obtaining metadata about the representation implied by the request without transferring the representation body. This method is often used for testing hypertext links for validity, accessibility, and recent modification.
The response to a HEAD request is cacheable and MAY be used to satisfy a subsequent HEAD request; see [Part6]. It also MAY be used to update a previously cached representation from that resource; if the new field values indicate that the cached representation differs from the current representation (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache MUST treat the cache entry as stale.

Bodies on HEAD requests have no defined semantics. Note that sending a body on a HEAD request might cause some existing implementations to reject the request.

7.5. POST

The POST method requests that the origin server accept the representation enclosed in the request as data to be processed by the target resource. POST is designed to allow a uniform method to cover the following functions:

- *Annotation of existing resources;
- *Posting a message to a bulletin board, newsgroup, mailing list, or similar group of articles;
- *Providing a block of data, such as the result of submitting a form, to a data-handling process;
- *Extending a database through an append operation.

The actual function performed by the POST method is determined by the server and is usually dependent on the effective request URI.
The action performed by the POST method might not result in a resource that can be identified by a URI. In this case, either 200 (OK) or 204 (No Content) is the appropriate response status code, depending on whether or not the response includes a representation that describes the result.
If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain a representation which describes the status of the request and refers to the new resource, and a Location header field (see Section 9.4).
Responses to POST requests are only cacheable when they include explicit freshness information (see Section 2.3.1 of [Part6]). A cached POST response with a Content-Location header field (see Section 6.7 of [Part3]) whose value is the effective Request URI MAY be used to satisfy subsequent GET and HEAD requests.
Note that POST caching is not widely implemented. However, the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource.

7.6. PUT

The PUT method requests that the state of the target resource be created or replaced with the state defined by the representation enclosed in the request message payload. A successful PUT of a given representation would suggest that a subsequent GET on that same target resource will result in an equivalent representation being returned in a 200 (OK) response. However, there is no guarantee that such a state change will be observable, since the target resource might be acted

upon by other user agents in parallel, or might be subject to dynamic processing by the origin server, before any subsequent GET is received. A successful response only implies that the user agent's intent was achieved at the time of its processing by the origin server.
If the target resource does not have a current representation and the PUT successfully creates one, then the origin server MUST inform the user agent by sending a 201 (Created) response. If the target resource does have a current representation and that representation is successfully modified in accordance with the state of the enclosed representation, then either a 200 (OK) or 204 (No Content) response SHOULD be sent to indicate successful completion of the request. Unrecognized header fields SHOULD be ignored (i.e., not saved as part of the resource state).
An origin server SHOULD verify that the PUT representation is consistent with any constraints which the server has for the target resource that cannot or will not be changed by the PUT. This is particularly important when the origin server uses internal configuration information related to the URI in order to set the values for representation metadata on GET responses. When a PUT representation is inconsistent with the target resource, the origin server SHOULD either make them consistent, by transforming the representation or changing the resource configuration, or respond with an appropriate error message containing sufficient information to explain why the representation is unsuitable. The 409 (Conflict) or 415 (Unsupported Media Type) status codes are suggested, with the latter being specific to constraints on Content-Type values.
For example, if the target resource is configured to always have a Content-Type of "text/html" and the representation being PUT has a Content-Type of "image/jpeg", then the origin server SHOULD do one of: (a) reconfigure the target resource to reflect the new media type; (b) transform the PUT representation to a format consistent with that of the resource before saving it as the new resource state; or, (c) reject the request with a 415 response indicating that the target resource is limited to "text/html", perhaps including a link to a different resource that would be a suitable target for the new representation.
HTTP does not define exactly how a PUT method affects the state of an origin server beyond what can be expressed by the intent of the user agent request and the semantics of the origin server response. It does not define what a resource might be, in any sense of that word, beyond the interface provided via HTTP. It does not define how resource state is "stored", nor how such storage might change as a result of a change in resource state, nor how the origin server translates resource state into representations. Generally speaking, all implementation details behind the resource interface are intentionally hidden by the server.
The fundamental difference between the POST and PUT methods is highlighted by the different intent for the target resource. The target resource in a POST request is intended to handle the enclosed representation as a data-accepting process, such as for a gateway to some other protocol or a document that accepts annotations. In

contrast, the target resource in a PUT request is intended to take the enclosed representation as a new or replacement value. Hence, the intent of PUT is idempotent and visible to intermediaries, even though the exact effect is only known by the origin server.

Proper interpretation of a PUT request presumes that the user agent knows what target resource is desired. A service that is intended to select a proper URI on behalf of the client, after receiving a state-changing request, SHOULD be implemented using the POST method rather than PUT. If the origin server will not make the requested PUT state change to the target resource and instead wishes to have it applied to a different resource, such as when the resource has been moved to a different URI, then the origin server MUST send a 301 (Moved Permanently) response; the user agent MAY then make its own decision regarding whether or not to redirect the request.

A PUT request applied to the target resource MAY have side-effects on other resources. For example, an article might have a URI for identifying "the current version" (a resource) which is separate from the URIs identifying each particular version (different resources that at one point shared the same state as the current version resource). A successful PUT request on "the current version" URI might therefore create a new version resource in addition to changing the state of the target resource, and might also cause links to be added between the related resources.

An origin server SHOULD reject any PUT request that contains a Content-Range header field, since it might be misinterpreted as partial content (or might be partial content that is being mistakenly PUT as a full representation). Partial content updates are possible by targeting a separately identified resource with state that overlaps a portion of the larger resource, or by using a different method that has been specifically defined for partial updates (for example, the PATCH method defined in [RFC5789]).

Responses to the PUT method are not cacheable. If a PUT request passes through a cache that has one or more stored responses for the effective request URI, those stored responses will be invalidated (see Section 2.5 of [Part6]).

7.7. DELETE

The DELETE method requests that the origin server delete the target resource. This method MAY be overridden by human intervention (or other means) on the origin server. The client cannot be guaranteed that the operation has been carried out, even if the status code returned from the origin server indicates that the action has been completed successfully. However, the server SHOULD NOT indicate success unless, at the time the response is given, it intends to delete the resource or move it to an inaccessible location.

A successful response SHOULD be 200 (OK) if the response includes an representation describing the status, 202 (Accepted) if the action has

not yet been enacted, or 204 (No Content) if the action has been enacted but the response does not include a representation.
Bodies on DELETE requests have no defined semantics. Note that sending a body on a DELETE request might cause some existing implementations to reject the request.
Responses to the DELETE method are not cacheable. If a DELETE request passes through a cache that has one or more stored responses for the effective request URI, those stored responses will be invalidated (see Section 2.5 of [Part6]).

7.8. TRACE

The TRACE method requests a remote, application-layer loop-back of the request message. The final recipient of the request SHOULD reflect the message received back to the client as the message-body of a 200 (OK) response. The final recipient is either the origin server or the first proxy to receive a Max-Forwards value of zero (0) in the request (see Section 9.5). A TRACE request MUST NOT include a message-body.
TRACE allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information. The value of the Via header field (Section 9.9 of [Part1]) is of particular interest, since it acts as a trace of the request chain. Use of the Max-Forwards header field allows the client to limit the length of the request chain, which is useful for testing a chain of proxies forwarding messages in an infinite loop.
If the request is valid, the response SHOULD have a Content-Type of "message/http" (see Section 10.3.1 of [Part1]) and contain a message-body that encloses a copy of the entire request message. Responses to the TRACE method are not cacheable.

7.9. CONNECT

The CONNECT method requests that the proxy establish a tunnel to the request-target and then restrict its behavior to blind forwarding of packets until the connection is closed.
When using CONNECT, the request-target MUST use the authority form (Section 4.1.2 of [Part1]); i.e., the request-target consists of only the host name and port number of the tunnel destination, separated by a colon. For example,

```
CONNECT server.example.com:80 HTTP/1.1
Host: server.example.com:80
```

Other HTTP mechanisms can be used normally with the CONNECT method — except end-to-end protocol Upgrade requests, since the tunnel must be established first.
For example, proxy authentication might be used to establish the authority to create a tunnel:

```
CONNECT server.example.com:80 HTTP/1.1
Host: server.example.com:80
Proxy-Authorization: basic aGVsbG86d29ybGQ=
```

Bodies on CONNECT requests have no defined semantics. Note that sending a body on a CONNECT request might cause some existing implementations to reject the request.
Like any other pipelined HTTP/1.1 request, data to be tunnel may be sent immediately after the blank line. The usual caveats also apply: data may be discarded if the eventual response is negative, and the connection may be reset with no response if more than one TCP segment is outstanding.

7.9.1. Establishing a Tunnel with CONNECT

Any successful (2xx) response to a CONNECT request indicates that the proxy has established a connection to the requested host and port, and has switched to tunneling the current connection to that server connection.
It may be the case that the proxy itself can only reach the requested origin server through another proxy. In this case, the first proxy SHOULD make a CONNECT request of that next proxy, requesting a tunnel to the authority. A proxy MUST NOT respond with any 2xx status code unless it has either a direct or tunnel connection established to the authority.
An origin server which receives a CONNECT request for itself MAY respond with a 2xx status code to indicate that a connection is established.
If at any point either one of the peers gets disconnected, any outstanding data that came from that peer will be passed to the other one, and after that also the other connection will be terminated by the proxy. If there is outstanding data to that peer undelivered, that data will be discarded.

8. Status Code Definitions

Each Status-Code is described below, including any metadata required in the response.

8.1. Informational 1xx

This class of status code indicates a provisional response, consisting only of the Status-Line and optional header fields, and is terminated by an empty line. There are no required header fields for this class of status code. Since HTTP/1.0 did not define any 1xx status codes, servers MUST NOT send a 1xx response to an HTTP/1.0 client except under experimental conditions.

A client MUST be prepared to accept one or more 1xx status responses prior to a regular response, even if the client does not expect a 100 (Continue) status message. Unexpected 1xx status responses MAY be ignored by a user agent.
Proxies MUST forward 1xx responses, unless the connection between the proxy and its client has been closed, or unless the proxy itself requested the generation of the 1xx response. (For example, if a proxy adds a "Expect: 100-continue" field when it forwards a request, then it need not forward the corresponding 100 (Continue) response(s).)

8.1.1. 100 Continue

The client SHOULD continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed. See Section 7.2.3 of [Part1] for detailed discussion of the use and handling of this status code.

8.1.2. 101 Switching Protocols

The server understands and is willing to comply with the client's request, via the Upgrade message header field (Section 9.8 of [Part1]), for a change in the application protocol being used on this connection. The server will switch protocols to those defined by the response's Upgrade header field immediately after the empty line which terminates the 101 response.
The protocol SHOULD be switched only when it is advantageous to do so. For example, switching to a newer version of HTTP is advantageous over older versions, and switching to a real-time, synchronous protocol might be advantageous when delivering resources that use such features.

8.2. Successful 2xx

This class of status code indicates that the client's request was successfully received, understood, and accepted.

8.2.1. 200 OK

The request has succeeded. The payload returned with the response is dependent on the method used in the request, for example:

GET a representation of the target resource is sent in the response;

HEAD the same representation as GET, except without the message-body;

POST a representation describing or containing the result of the action;

TRACE
: a representation containing the request message as received by the end server.

Caches MAY use a heuristic (see Section 2.3.1.1 of [Part6]) to determine freshness for 200 responses.

8.2.2. 201 Created

The request has been fulfilled and has resulted in a new resource being created. The newly created resource can be referenced by the URI(s) returned in the payload of the response, with the most specific URI for the resource given by a Location header field. The response SHOULD include a payload containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The payload format is specified by the media type given in the Content-Type header field. The origin server MUST create the resource before returning the 201 status code. If the action cannot be carried out immediately, the server SHOULD respond with 202 (Accepted) response instead.
A 201 response MAY contain an ETag response header field indicating the current value of the entity-tag for the representation of the resource just created (see Section 2.2 of [Part4]).

8.2.3. 202 Accepted

The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.
The 202 response is intentionally non-committal. Its purpose is to allow a server to accept a request for some other process (perhaps a batch-oriented process that is only run once per day) without requiring that the user agent's connection to the server persist until the process is completed. The representation returned with this response SHOULD include an indication of the request's current status and either a pointer to a status monitor or some estimate of when the user can expect the request to be fulfilled.

8.2.4. 203 Non-Authoritative Information

The returned metadata in the header fields is not the definitive set as available from the origin server, but is gathered from a local or a third-party copy. The set presented MAY be a subset or superset of the original version. For example, including local annotation information about the resource might result in a superset of the metadata known by the origin server. Use of this response code is not required and is only appropriate when the response would otherwise be 200 (OK).

Caches MAY use a heuristic (see Section 2.3.1.1 of [Part6]) to determine freshness for 203 responses.

8.2.5. 204 No Content

The 204 (No Content) status code indicates that the server has successfully fulfilled the request and that there is no additional content to return in the response payload body. Metadata in the response header fields refer to the target resource and its current representation after the requested action.
For example, if a 204 status code is received in response to a PUT request and the response contains an ETag header field, then the PUT was successful and the ETag field-value contains the entity-tag for the new representation of that target resource.
The 204 response allows a server to indicate that the action has been successfully applied to the target resource while implying that the user agent SHOULD NOT traverse away from its current "document view" (if any). The server assumes that the user agent will provide some indication of the success to its user, in accord with its own interface, and apply any new or updated metadata in the response to the active representation. For example, a 204 status code is commonly used with document editing interfaces corresponding to a "save" action, such that the document being saved remains available to the user for editing. It is also frequently used with interfaces that expect automated data transfers to be prevalent, such as within distributed version control systems.
The 204 response MUST NOT include a message-body, and thus is always terminated by the first empty line after the header fields.

8.2.6. 205 Reset Content

The server has fulfilled the request and the user agent SHOULD reset the document view which caused the request to be sent. This response is primarily intended to allow input for actions to take place via user input, followed by a clearing of the form in which the input is given so that the user can easily initiate another input action.
The message-body included with the response MUST be empty. Note that receivers still need to parse the response according to the algorithm defined in Section 3.3 of [Part1].

8.2.7. 206 Partial Content

The server has fulfilled the partial GET request for the resource and the enclosed payload is a partial representation as defined in Section 3.1 of [Part5].
Caches MAY use a heuristic (see Section 2.3.1.1 of [Part6]) to determine freshness for 206 responses.

8.3. Redirection 3xx

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required MAY be carried out by the user agent without interaction with the user if and only if the method used in the second request is known to be "safe", as defined in Section 7.1.1. A client SHOULD detect infinite redirection loops, since such loops generate network traffic for each redirection.

> *Note: An earlier version of this specification recommended a maximum of five redirections ([RFC2068], Section 10.3). Content developers need to be aware that some clients might implement such a fixed limitation.

8.3.1. 300 Multiple Choices

The target resource has more than one representation, each with its own specific location, and agent-driven negotiation information (Section 5 of [Part3]) is being provided so that the user (or user agent) can select a preferred representation by redirecting its request to that location.
Unless it was a HEAD request, the response SHOULD include a representation containing a list of representation metadata and location(s) from which the user or user agent can choose the one most appropriate. The data format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.
If the server has a preferred choice of representation, it SHOULD include the specific URI for that representation in the Location field; user agents MAY use the Location field value for automatic redirection.
Caches MAY use a heuristic (see Section 2.3.1.1 of [Part6]) to determine freshness for 300 responses.

8.3.2. 301 Moved Permanently

The target resource has been assigned a new permanent URI and any future references to this resource SHOULD use one of the returned URIs. Clients with link editing capabilities ought to automatically re-link references to the effective request URI to one or more of the new references returned by the server, where possible.
Caches MAY use a heuristic (see Section 2.3.1.1 of [Part6]) to determine freshness for 301 responses.
The new permanent URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the representation of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 301 status code is received in response to a request method that is known to be "safe", as defined in Section 7.1.1, then the request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

> *Note: When automatically redirecting a POST request after receiving a 301 status code, some existing HTTP/1.0 user agents will erroneously change it into a GET request.

8.3.3. 302 Found

The target resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client SHOULD continue to use the effective request URI for future requests.
The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the representation of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).
If the 302 status code is received in response to a request method that is known to be "safe", as defined in Section 7.1.1, then the request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

> *Note: HTTP/1.0 ([RFC1945], Section 9.3) and the first version of HTTP/1.1 ([RFC2068], Section 10.3.3) specify that the client is not allowed to change the method on the redirected request. However, most existing user agent implementations treat 302 as if it were a 303 response, performing a GET on the Location field-value regardless of the original request method. Therefore, a previous version of this specification ([RFC2616], Section 10.3.3) has added the status codes 303 *[status.303]* and 307 *[status.307]* for servers that wish to make unambiguously clear which kind of reaction is expected of the client.

8.3.4. 303 See Other

The server directs the user agent to a different resource, indicated by a URI in the Location header field, that provides an indirect response to the original request. The user agent MAY perform a GET request on the URI in the Location field in order to obtain a representation corresponding to the response, be redirected again, or end with an error status. The Location URI is not a substitute reference for the effective request URI.
The 303 status code is generally applicable to any HTTP method. It is primarily used to allow the output of a POST action to redirect the

user agent to a selected resource, since doing so provides the information corresponding to the POST response in a form that can be separately identified, bookmarked, and cached independent of the original request.
A 303 response to a GET request indicates that the requested resource does not have a representation of its own that can be transferred by the server over HTTP. The Location URI indicates a resource that is descriptive of the target resource, such that the follow-on representation might be useful to recipients without implying that it adequately represents the target resource. Note that answers to the questions of what can be represented, what representations are adequate, and what might be a useful description are outside the scope of HTTP and thus entirely determined by the URI owner(s).
Except for responses to a HEAD request, the representation of a 303 response SHOULD contain a short hypertext note with a hyperlink to the Location URI.

8.3.5. 304 Not Modified

The response to the request has not been modified since the conditions indicated by the client's conditional GET request, as defined in Section 4.1 of [Part4].

8.3.6. 305 Use Proxy

The 305 status code was defined in a previous version of this specification (see Appendix Appendix A), and is now deprecated.

8.3.7. 306 (Unused)

The 306 status code was used in a previous version of the specification, is no longer used, and the code is reserved.

8.3.8. 307 Temporary Redirect

The target resource resides temporarily under a different URI. Since the redirection can change over time, the client SHOULD continue to use the effective request URI for future requests.
The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the representation of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s), since many pre-HTTP/1.1 user agents do not understand the 307 status code. Therefore, the note SHOULD contain the information necessary for a user to repeat the original request on the new URI.
If the 307 status code is received in response to a request method that is known to be "safe", as defined in Section 7.1.1, then the request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

8.4. Client Error 4xx

The 4xx class of status code is intended for cases in which the client seems to have erred. Except when responding to a HEAD request, the server SHOULD include a representation containing an explanation of the error situation, and whether it is a temporary or permanent condition. These status codes are applicable to any request method. User agents SHOULD display any included representation to the user.
If the client is sending data, a server implementation using TCP SHOULD be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which might erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.

8.4.1. 400 Bad Request

The request could not be understood by the server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.

8.4.2. 401 Unauthorized

The request requires user authentication (see Section 3.1 of [Part7]).

8.4.3. 402 Payment Required

This code is reserved for future use.

8.4.4. 403 Forbidden

The server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated. If the request method was not HEAD and the server wishes to make public why the request has not been fulfilled, it SHOULD describe the reason for the refusal in the representation. If the server does not wish to make this information available to the client, the status code 404 (Not Found) can be used instead.

8.4.5. 404 Not Found

The server has not found anything matching the effective request URI. No indication is given of whether the condition is temporary or permanent. The 410 (Gone) status code SHOULD be used if the server knows, through some internally configurable mechanism, that an old resource is permanently unavailable and has no forwarding address. This status code is commonly used when the server does not wish to reveal exactly why the request has been refused, or when no other response is applicable.

8.4.6. 405 Method Not Allowed

The method specified in the Request-Line is not allowed for the target resource. The response MUST include an Allow header field containing a list of valid methods for the requested resource.

8.4.7. 406 Not Acceptable

The resource identified by the request is only capable of generating response representations which have content characteristics not acceptable according to the accept header fields sent in the request. Unless it was a HEAD request, the response SHOULD include a representation containing a list of available representation characteristics and location(s) from which the user or user agent can choose the one most appropriate. The data format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

> *Note: HTTP/1.1 servers are allowed to return responses which are not acceptable according to the accept header fields sent in the request. In some cases, this might even be preferable to sending a 406 response. User agents are encouraged to inspect the header fields of an incoming response to determine if it is acceptable.

If the response could be unacceptable, a user agent SHOULD temporarily stop receipt of more data and query the user for a decision on further actions.

8.4.8. 407 Proxy Authentication Required

This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy (see Section 3.2 of [Part7]).

8.4.9. 408 Request Timeout

The client did not produce a request within the time that the server was prepared to wait. The client MAY repeat the request without modifications at any later time.

8.4.10. 409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough information for the user to recognize the source of the conflict.

Ideally, the response representation would include enough information for the user or user agent to fix the problem; however, that might not be possible and is not required.
Conflicts are most likely to occur in response to a PUT request. For example, if versioning were being used and the representation being PUT included changes to a resource which conflict with those made by an earlier (third-party) request, the server might use the 409 response to indicate that it can't complete the request. In this case, the response representation would likely contain a list of the differences between the two versions in a format defined by the response Content-Type.

8.4.11. 410 Gone

The target resource is no longer available at the server and no forwarding address is known. This condition is expected to be considered permanent. Clients with link editing capabilities SHOULD delete references to the effective request URI after user approval. If the server does not know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) SHOULD be used instead.
The 410 response is primarily intended to assist the task of web maintenance by notifying the recipient that the resource is intentionally unavailable and that the server owners desire that remote links to that resource be removed. Such an event is common for limited-time, promotional services and for resources belonging to individuals no longer working at the server's site. It is not necessary to mark all permanently unavailable resources as "gone" or to keep the mark for any length of time — that is left to the discretion of the server owner.
Caches MAY use a heuristic (see Section 2.3.1.1 of [Part6]) to determine freshness for 410 responses.

8.4.12. 411 Length Required

The server refuses to accept the request without a defined Content-Length. The client MAY repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.

8.4.13. 412 Precondition Failed

The precondition given in one or more of the header fields evaluated to false when it was tested on the server, as defined in Section 4.2 of [Part4].

8.4.14. 413 Request Entity Too Large

The server is refusing to process a request because the request representation is larger than the server is willing or able to process. The server MAY close the connection to prevent the client from continuing the request.

If the condition is temporary, the server SHOULD include a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

8.4.15. 414 URI Too Long

The server is refusing to service the request because the effective request URI is longer than the server is willing to interpret. This rare condition is only likely to occur when a client has improperly converted a POST request to a GET request with long query information, when the client has descended into a URI "black hole" of redirection (e.g., a redirected URI prefix that points to a suffix of itself), or when the server is under attack by a client attempting to exploit security holes present in some servers using fixed-length buffers for reading or manipulating the effective request URI.

8.4.16. 415 Unsupported Media Type

The server is refusing to service the request because the request payload is in a format not supported by this request method on the target resource.

8.4.17. 416 Requested Range Not Satisfiable

The request included a Range header field (Section 5.4 of [Part5]) and none of the range-specifier values in this field overlap the current extent of the selected resource. See Section 3.2 of [Part5].

8.4.18. 417 Expectation Failed

The expectation given in an Expect header field (see Section 9.2) could not be met by this server, or, if the server is a proxy, the server has unambiguous evidence that the request could not be met by the next-hop server.

8.4.19. 426 Upgrade Required

The request can not be completed without a prior protocol upgrade. This response MUST include an Upgrade header field (Section 9.8 of [Part1]) specifying the required protocols.
Example:

```
HTTP/1.1 426 Upgrade Required
Upgrade: HTTP/2.0
Connection: Upgrade
```

The server SHOULD include a message body in the 426 response which indicates in human readable form the reason for the error and describes any alternative courses which may be available to the user.

8.5. Server Error 5xx

Response status codes beginning with the digit "5" indicate cases in which the server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the server SHOULD include a representation containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included representation to the user. These response codes are applicable to any request method.

8.5.1. 500 Internal Server Error

The server encountered an unexpected condition which prevented it from fulfilling the request.

8.5.2. 501 Not Implemented

The server does not support the functionality required to fulfill the request. This is the appropriate response when the server does not recognize the request method and is not capable of supporting it for any resource.

8.5.3. 502 Bad Gateway

The server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request.

8.5.4. 503 Service Unavailable

The server is currently unable to handle the request due to a temporary overloading or maintenance of the server. The implication is that this is a temporary condition which will be alleviated after some delay. If known, the length of the delay MAY be indicated in a Retry-After header field. If no Retry-After is given, the client SHOULD handle the response as it would for a 500 response.

> *Note: The existence of the 503 status code does not imply that a server must use it when becoming overloaded. Some servers might wish to simply refuse the connection.

8.5.5. 504 Gateway Timeout

The server, while acting as a gateway or proxy, did not receive a timely response from the upstream server specified by the URI (e.g.,

HTTP, FTP, LDAP) or some other auxiliary server (e.g., DNS) it needed to access in attempting to complete the request.

*Note to implementors: some deployed proxies are known to return 400 or 500 when DNS lookups time out.

8.5.6. 505 HTTP Version Not Supported

The server does not support, or refuses to support, the protocol version that was used in the request message. The server is indicating that it is unable or unwilling to complete the request using the same major version as the client, as described in Section 2.5 of [Part1], other than with this error message. The response SHOULD contain a representation describing why that version is not supported and what other protocols are supported by that server.

9. Header Field Definitions

This section defines the syntax and semantics of HTTP/1.1 header fields related to request and response semantics.

9.1. Allow

The "Allow" header field lists the set of methods advertised as supported by the target resource. The purpose of this field is strictly to inform the recipient of valid request methods associated with the resource.

```
Allow = #Method
```

Example of use:

```
Allow: GET, HEAD, PUT
```

The actual set of allowed methods is defined by the origin server at the time of each request.
A proxy MUST NOT modify the Allow header field – it does not need to understand all the methods specified in order to handle them according to the generic message handling rules.

9.2. Expect

The "Expect" header field is used to indicate that particular server behaviors are required by the client.

```
  Expect       = 1#expectation

  expectation  = "100-continue" / expectation-extension
  expectation-extension = token [ "=" ( token / quoted-string )
                          *expect-params ]
  expect-params = ";" token [ "=" ( token / quoted-string ) ]
```

A server that does not understand or is unable to comply with any of the expectation values in the Expect field of a request MUST respond with appropriate error status code. The server MUST respond with a 417 (Expectation Failed) status code if any of the expectations cannot be met or, if there are other problems with the request, some other 4xx status code.

This header field is defined with extensible syntax to allow for future extensions. If a server receives a request containing an Expect field that includes an expectation-extension that it does not support, it MUST respond with a 417 (Expectation Failed) status code.

Comparison of expectation values is case-insensitive for unquoted tokens (including the 100-continue token), and is case-sensitive for quoted-string expectation-extensions.

The Expect mechanism is hop-by-hop: that is, an HTTP/1.1 proxy MUST return a 417 (Expectation Failed) status code if it receives a request with an expectation that it cannot meet. However, the Expect header field itself is end-to-end; it MUST be forwarded if the request is forwarded.

Many older HTTP/1.0 and HTTP/1.1 applications do not understand the Expect header field.

See Section 7.2.3 of [Part1] for the use of the 100 (Continue) status code.

9.3. From

The "From" header field, if given, SHOULD contain an Internet e-mail address for the human user who controls the requesting user agent. The address SHOULD be machine-usable, as defined by "mailbox" in Section 3.4 of [RFC5322]:

```
  From    = mailbox

  mailbox = <mailbox, defined in [RFC5322], Section 3.4>
```

An example is:

```
  From: webmaster@example.org
```

This header field MAY be used for logging purposes and as a means for identifying the source of invalid or unwanted requests. It SHOULD NOT be used as an insecure form of access protection. The interpretation of this field is that the request is being performed on behalf of the person given, who accepts responsibility for the method performed. In

particular, robot agents SHOULD include this header field so that the person responsible for running the robot can be contacted if problems occur on the receiving end.
The Internet e-mail address in this field MAY be separate from the Internet host which issued the request. For example, when a request is passed through a proxy the original issuer's address SHOULD be used.
The client SHOULD NOT send the From header field without the user's approval, as it might conflict with the user's privacy interests or their site's security policy. It is strongly recommended that the user be able to disable, enable, and modify the value of this field at any time prior to a request.

9.4. Location

The "Location" header field is used to identify a newly created resource, or to redirect the recipient to a different location for completion of the request.
For 201 (Created) responses, the Location is the URI of the new resource which was created by the request. For 3xx responses, the location SHOULD indicate the server's preferred URI for automatic redirection to the resource.
The field value consists of a single URI-reference. When it has the form of a relative reference ([RFC3986], Section 4.2), the final value is computed by resolving it against the effective request URI ([RFC3986], Section 5).

```
  Location = URI-reference
```

Examples are:

```
  Location: http://www.example.org/pub/WWW/People.html#tim
```

```
  Location: /index.html
```

There are circumstances in which a fragment identifier in a Location URI would not be appropriate:

- With a 201 Created response, because in this usage the Location header field specifies the URI for the entire created resource.

- With 305 Use Proxy.

- Note: This specification does not define precedence rules for the case where the original URI, as navigated to by the user agent, and the Location header field value both contain fragment identifiers. Thus be aware that including fragment identifiers might inconvenience anyone relying on the semantics of the original URI's fragment identifier.

*Note: The Content-Location header field (Section 6.7 of [Part3]) differs from Location in that the Content-Location identifies the most specific resource corresponding to the enclosed representation. It is therefore possible for a response to contain header fields for both Location and Content-Location.

9.5. Max-Forwards

The "Max-Forwards" header field provides a mechanism with the TRACE (Section 7.8) and OPTIONS (Section 7.2) methods to limit the number of times that the request is forwarded by proxies. This can be useful when the client is attempting to trace a request which appears to be failing or looping in mid-chain.

```
Max-Forwards = 1*DIGIT
```

The Max-Forwards value is a decimal integer indicating the remaining number of times this request message can be forwarded.
Each recipient of a TRACE or OPTIONS request containing a Max-Forwards header field MUST check and update its value prior to forwarding the request. If the received value is zero (0), the recipient MUST NOT forward the request; instead, it MUST respond as the final recipient.
If the received Max-Forwards value is greater than zero, then the forwarded message MUST contain an updated Max-Forwards field with a value decremented by one (1).
The Max-Forwards header field MAY be ignored for all other request methods.

9.6. Referer

The "Referer" [sic] header field allows the client to specify the URI of the resource from which the effective request URI was obtained (the "referrer", although the header field is misspelled.).
The Referer header field allows servers to generate lists of back-links to resources for interest, logging, optimized caching, etc. It also allows obsolete or mistyped links to be traced for maintenance. Some servers use Referer as a means of controlling where they allow links from (so-called "deep linking"), but legitimate requests do not always contain a Referer header field.
If the effective request URI was obtained from a source that does not have its own URI (e.g., input from the user keyboard), the Referer field MUST either be sent with the value "about:blank", or not be sent at all. Note that this requirement does not apply to sources with non-HTTP URIs (e.g., FTP).

```
Referer = absolute-URI / partial-URI
```

Example:

```
  Referer: http://www.example.org/hypertext/Overview.html
```

If the field value is a relative URI, it SHOULD be interpreted relative to the effective request URI. The URI MUST NOT include a fragment. See Section 11.2 for security considerations.

9.7. Retry-After

The header "Retry-After" field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client. This field MAY also be used with any 3xx (Redirection) response to indicate the minimum time the user-agent is asked wait before issuing the redirected request.
The value of this field can be either an HTTP-date or an integer number of seconds (in decimal) after the time of the response.

```
  Retry-After = HTTP-date / delta-seconds
```

Time spans are non-negative decimal integers, representing time in seconds.

```
  delta-seconds  = 1*DIGIT
```

Two examples of its use are

```
  Retry-After: Fri, 31 Dec 1999 23:59:59 GMT
  Retry-After: 120
```

In the latter example, the delay is 2 minutes.

9.8. Server

The "Server" header field contains information about the software used by the origin server to handle the request.
The field can contain multiple product tokens (Section 6.3 of [Part1]) and comments (Section 3.2 of [Part1]) identifying the server and any significant subproducts. The product tokens are listed in order of their significance for identifying the application.

```
  Server = product *( RWS ( product / comment ) )
```

Example:

```
  Server: CERN/3.0 libwww/2.17
```

If the response is being forwarded through a proxy, the proxy application MUST NOT modify the Server header field. Instead, it MUST include a Via field (as described in Section 9.9 of [Part1]).

*Note: Revealing the specific software version of the server might allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Server implementors are encouraged to make this field a configurable option.

9.9. User-Agent

The "User-Agent" header field contains information about the user agent originating the request. User agents SHOULD include this field with requests.
Typically, it is used for statistical purposes, the tracing of protocol violations, and tailoring responses to avoid particular user agent limitations.
The field can contain multiple product tokens (Section 6.3 of [Part1]) and comments (Section 3.2 of [Part1]) identifying the agent and its significant subproducts. By convention, the product tokens are listed in order of their significance for identifying the application.
Because this field is usually sent on every request a user agent makes, implementations are encouraged not to include needlessly fine-grained detail, and to limit (or even prohibit) the addition of subproducts by third parties. Overly long and detailed User-Agent field values make requests larger and can also be used to identify ("fingerprint") the user against their wishes.
Likewise, implementations are encouraged not to use the product tokens of other implementations in order to declare compatibility with them, as this circumvents the purpose of the field. Finally, they are encouraged not to use comments to identify products; doing so makes the field value more difficult to parse.

```
  User-Agent = product *( RWS ( product / comment ) )
```

Example:

```
  User-Agent: CERN-LineMode/2.15 libwww/2.17b3
```

10. IANA Considerations

10.1. Method Registry

The registration procedure for HTTP request methods is defined by Section 2.2 of this document.
The HTTP Method Registry shall be created at http://www.iana.org/assignments/http-methods and be populated with the registrations below:

Method	Safe	Reference
CONNECT	no	Section 7.9
DELETE	no	Section 7.7

Method	Safe	Reference
GET	yes	Section 7.3
HEAD	yes	Section 7.4
OPTIONS	yes	Section 7.2
POST	no	Section 7.5
PUT	no	Section 7.6
TRACE	yes	Section 7.8

10.2. Status Code Registry

The registration procedure for HTTP Status Codes — previously defined in Section 7.1 of [RFC2817] — is now defined by Section 4.2 of this document.

The HTTP Status Code Registry located at http://www.iana.org/assignments/http-status-codes shall be updated with the registrations below:

Value	Description	Reference
100	Continue	Section 8.1.1
101	Switching Protocols	Section 8.1.2
200	OK	Section 8.2.1
201	Created	Section 8.2.2
202	Accepted	Section 8.2.3
203	Non-Authoritative Information	Section 8.2.4
204	No Content	Section 8.2.5
205	Reset Content	Section 8.2.6
300	Multiple Choices	Section 8.3.1
301	Moved Permanently	Section 8.3.2
302	Found	Section 8.3.3
303	See Other	Section 8.3.4
305	Use Proxy	Section 8.3.6
306	(Unused)	Section 8.3.7
307	Temporary Redirect	Section 8.3.8
400	Bad Request	Section 8.4.1
402	Payment Required	Section 8.4.3
403	Forbidden	Section 8.4.4
404	Not Found	Section 8.4.5
405	Method Not Allowed	Section 8.4.6

Value	Description	Reference
406	Not Acceptable	Section 8.4.7
407	Proxy Authentication Required	Section 8.4.8
408	Request Timeout	Section 8.4.9
409	Conflict	Section 8.4.10
410	Gone	Section 8.4.11
411	Length Required	Section 8.4.12
413	Request Entity Too Large	Section 8.4.14
414	URI Too Long	Section 8.4.15
415	Unsupported Media Type	Section 8.4.16
417	Expectation Failed	Section 8.4.18
426	Upgrade Required	Section 8.4.19
500	Internal Server Error	Section 8.5.1
501	Not Implemented	Section 8.5.2
502	Bad Gateway	Section 8.5.3
503	Service Unavailable	Section 8.5.4
504	Gateway Timeout	Section 8.5.5
505	HTTP Version Not Supported	Section 8.5.6

10.3. Header Field Registration

The Message Header Field Registry located at http://www.iana.org/assignments/message-headers/message-header-index.html shall be updated with the permanent registrations below (see [RFC3864]):

Header Field Name	Protocol	Status	Reference
Allow	http	standard	Section 9.1
Expect	http	standard	Section 9.2
From	http	standard	Section 9.3
Location	http	standard	Section 9.4
Max-Forwards	http	standard	Section 9.5
Referer	http	standard	Section 9.6
Retry-After	http	standard	Section 9.7
Server	http	standard	Section 9.8
User-Agent	http	standard	Section 9.9

The change controller is: "IETF (iesg@ietf.org) - Internet Engineering Task Force".

11. Security Considerations

This section is meant to inform application developers, information providers, and users of the security limitations in HTTP/1.1 as described by this document. The discussion does not include definitive solutions to the problems revealed, though it does make some suggestions for reducing security risks.

11.1. Transfer of Sensitive Information

Like any generic data transfer protocol, HTTP cannot regulate the content of the data that is transferred, nor is there any a priori method of determining the sensitivity of any particular piece of information within the context of any given request. Therefore, applications SHOULD supply as much control over this information as possible to the provider of that information. Four header fields are worth special mention in this context: Server, Via, Referer and From. Revealing the specific software version of the server might allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Implementors SHOULD make the Server header field a configurable option.

Proxies which serve as a portal through a network firewall SHOULD take special precautions regarding the transfer of header information that identifies the hosts behind the firewall. In particular, they SHOULD remove, or replace with sanitized versions, any Via fields generated behind the firewall.

The Referer header field allows reading patterns to be studied and reverse links drawn. Although it can be very useful, its power can be abused if user details are not separated from the information contained in the Referer. Even when the personal information has been removed, the Referer header field might indicate a private document's URI whose publication would be inappropriate.

The information sent in the From field might conflict with the user's privacy interests or their site's security policy, and hence it SHOULD NOT be transmitted without the user being able to disable, enable, and modify the contents of the field. The user MUST be able to set the contents of this field within a user preference or application defaults configuration.

We suggest, though do not require, that a convenient toggle interface be provided for the user to enable or disable the sending of From and Referer information.

The User-Agent (Section 9.9) or Server (Section 9.8) header fields can sometimes be used to determine that a specific client or server have a particular security hole which might be exploited. Unfortunately, this same information is often used for other valuable purposes for which HTTP currently has no better mechanism.

Furthermore, the User-Agent header field may contain enough entropy to be used, possibly in conjunction with other material, to uniquely identify the user.

Some request methods, like TRACE (Section 7.8), expose information that was sent in request header fields within the body of their response. Clients SHOULD be careful with sensitive information, like Cookies, Authorization credentials, and other header fields that might be used to collect data from the client.

11.2. Encoding Sensitive Information in URIs

Because the source of a link might be private information or might reveal an otherwise private information source, it is strongly recommended that the user be able to select whether or not the Referer field is sent. For example, a browser client could have a toggle switch for browsing openly/anonymously, which would respectively enable/ disable the sending of Referer and From information.
Clients SHOULD NOT include a Referer header field in a (non-secure) HTTP request if the referring page was transferred with a secure protocol.
Authors of services SHOULD NOT use GET-based forms for the submission of sensitive data because that data will be placed in the request-target. Many existing servers, proxies, and user agents log or display the request-target in places where it might be visible to third parties. Such services can use POST-based form submission instead.

11.3. Location Headers and Spoofing

If a single server supports multiple organizations that do not trust one another, then it MUST check the values of Location and Content-Location header fields in responses that are generated under control of said organizations to make sure that they do not attempt to invalidate resources over which they have no authority.

11.4. Security Considerations for CONNECT

Since tunneled data is opaque to the proxy, there are additional risks to tunneling to other well-known or reserved ports. A HTTP client CONNECTing to port 25 could relay spam via SMTP, for example. As such, proxies SHOULD restrict CONNECT access to a small number of known ports.

12. Acknowledgments

13. References

13.1. Normative References

[Part1]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 1: URIs, Connections, and Message Parsing", Internet-Draft draft-ietf-httpbis-p1-messaging-14, April 2011.

[Part3]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 3: Message Payload and Content Negotiation", Internet-Draft draft-ietf-httpbis-p3-payload-14, April 2011.
[Part4]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 4: Conditional Requests", Internet-Draft draft-ietf-httpbis-p4-conditional-14, April 2011.
[Part5]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 5: Range Requests and Partial Responses", Internet-Draft draft-ietf-httpbis-p5-range-14, April 2011.
[Part6]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Nottingham, M. and J. F. Reschke, "HTTP/1.1, part 6: Caching", Internet-Draft draft-ietf-httpbis-p6-cache-14, April 2011.
[Part7]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y. and J. F. Reschke, "HTTP/1.1, part 7: Authentication", Internet-Draft draft-ietf-httpbis-p7-auth-14, April 2011.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
[RFC3986]	Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
[RFC5234]	Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

13.2. Informative References

[RFC1945]	Berners-Lee, T., Fielding, R.T. and H.F. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.
[RFC2068]	Fielding, R., Gettys, J., Mogul, J., Nielsen, H. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 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.
[RFC2817]	Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.
[RFC3864]	

	Klyne, G., Nottingham, M. and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.
[RFC5226]	Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
[RFC5322]	Resnick, P., "Internet Message Format", RFC 5322, October 2008.
[RFC5789]	Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

Appendix A. Changes from RFC 2616

This document takes over the Status Code Registry, previously defined in Section 7.1 of [RFC2817]. (Section 4.2)
Clarify definition of POST. (Section 7.5)
Remove requirement to handle all Content-* header fields; ban use of Content-Range with PUT. (Section 7.6)
Take over definition of CONNECT method from [RFC2817]. (Section 7.9)
Failed to consider that there are many other request methods that are safe to automatically redirect, and further that the user agent is able to make that determination based on the request method semantics. (Sections 8.3.2, 8.3.3 and 8.3.8)
Deprecate 305 Use Proxy status code, because user agents did not implement it. It used to indicate that the target resource must be accessed through the proxy given by the Location field. The Location field gave the URI of the proxy. The recipient was expected to repeat this single request via the proxy. (Section 8.3.6)
Define status 426 (Upgrade Required) (this was incorporated from [RFC2817]). (Section 8.4.19)
Change ABNF productions for header fields to only define the field value. (Section 9)
Reclassify "Allow" as response header field, removing the option to specify it in a PUT request. Relax the server requirement on the contents of the Allow header field and remove requirement on clients to always trust the header field value. (Section 9.1)
Correct syntax of Location header field to allow URI references (including relative references and fragments), as referred symbol "absoluteURI" wasn't what was expected, and add some clarifications as to when use of fragments would not be appropriate. (Section 9.4)
Restrict Max-Forwards header field to OPTIONS and TRACE (previously, extension methods could have used it as well). (Section 9.5)
Allow Referer field value of "about:blank" as alternative to not specifying it. (Section 9.6)
In the description of the Server header field, the Via field was described as a SHOULD. The requirement was and is stated correctly in the description of the Via header field in Section 9.9 of [Part1]. (Section 9.8)

Appendix B. Collected ABNF

```
Allow = [ ( "," / Method ) *( OWS "," [ OWS Method ] ) ]

Expect = *( "," OWS ) expectation *( OWS "," [ OWS expectation ] )

From = mailbox

HTTP-date = <HTTP-date, defined in [Part1], Section 6.1>

Location = URI-reference

Max-Forwards = 1*DIGIT
Method = token

OWS = <OWS, defined in [Part1], Section 1.2.2>

RWS = <RWS, defined in [Part1], Section 1.2.2>
Reason-Phrase = *( WSP / VCHAR / obs-text )
Referer = absolute-URI / partial-URI
Retry-After = HTTP-date / delta-seconds

Server = product *( RWS ( product / comment ) )
Status-Code = 3DIGIT

URI-reference = <URI-reference, defined in [Part1], Section 2.6>
User-Agent = product *( RWS ( product / comment ) )

absolute-URI = <absolute-URI, defined in [Part1], Section 2.6>

comment = <comment, defined in [Part1], Section 3.2>

delta-seconds = 1*DIGIT

expect-params = ";" token [ "=" ( token / quoted-string ) ]
expectation = "100-continue" / expectation-extension
expectation-extension = token [ "=" ( token / quoted-string )
 *expect-params ]

mailbox = <mailbox, defined in [RFC5322], Section 3.4>

obs-text = <obs-text, defined in [Part1], Section 1.2.2>

partial-URI = <partial-URI, defined in [Part1], Section 2.6>
product = <product, defined in [Part1], Section 6.3>

quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>

token = <token, defined in [Part1], Section 1.2.2>

ABNF diagnostics:
```

```
; Allow defined but not used
; Expect defined but not used
; From defined but not used
; Location defined but not used
; Max-Forwards defined but not used
; Reason-Phrase defined but not used
; Referer defined but not used
; Retry-After defined but not used
; Server defined but not used
; Status-Code defined but not used
; User-Agent defined but not used
```

Appendix C. Change Log (to be removed by RFC Editor before publication)

Appendix C.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

Appendix C.2. Since draft-ietf-httpbis-p2-semantics-00

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/5: "Via is a MUST" (http://purl.org/NET/http-errata#via-must)

* http://tools.ietf.org/wg/httpbis/trac/ticket/6: "Fragments allowed in Location" (http://purl.org/NET/http-errata#location-fragments)

* http://tools.ietf.org/wg/httpbis/trac/ticket/10: "Safe Methods vs Redirection" (http://purl.org/NET/http-errata#saferedirect)

* http://tools.ietf.org/wg/httpbis/trac/ticket/17: "Revise description of the POST method" (http://purl.org/NET/http-errata#post)

* http://tools.ietf.org/wg/httpbis/trac/ticket/35: "Normative and Informative references"

* http://tools.ietf.org/wg/httpbis/trac/ticket/42: "RFC2606 Compliance"

* http://tools.ietf.org/wg/httpbis/trac/ticket/65: "Informative references"

* http://tools.ietf.org/wg/httpbis/trac/ticket/84: "Redundant cross-references"

Other changes:

* Move definitions of 304 and 412 condition codes to [Part4]

Appendix C.3. Since draft-ietf-httpbis-p2-semantics-01

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/21: "PUT side effects"

* http://tools.ietf.org/wg/httpbis/trac/ticket/91: "Duplicate Host header requirements"

Ongoing work on ABNF conversion (http://tools.ietf.org/wg/httpbis/trac/ticket/36):

* Move "Product Tokens" section (back) into Part 1, as "token" is used in the definition of the Upgrade header field.

* Add explicit references to BNF syntax and rules imported from other parts of the specification.

* Copy definition of delta-seconds from Part6 instead of referencing it.

Appendix C.4. Since draft-ietf-httpbis-p2-semantics-02

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/24: "Requiring Allow in 405 responses"

* http://tools.ietf.org/wg/httpbis/trac/ticket/59: "Status Code Registry"

* http://tools.ietf.org/wg/httpbis/trac/ticket/61: "Redirection vs. Location"

* http://tools.ietf.org/wg/httpbis/trac/ticket/70: "Cacheability of 303 response"

* http://tools.ietf.org/wg/httpbis/trac/ticket/76: "305 Use Proxy"

* http://tools.ietf.org/wg/httpbis/trac/ticket/105: "Classification for Allow header"

* http://tools.ietf.org/wg/httpbis/trac/ticket/112: "PUT - 'store under' vs 'store at'"

Ongoing work on IANA Message Header Field Registration (http://tools.ietf.org/wg/httpbis/trac/ticket/40):

*Reference RFC 3984, and update header field registrations for headers defined in this document.

Ongoing work on ABNF conversion (http://tools.ietf.org/wg/httpbis/trac/ticket/36):

*Replace string literals when the string really is case-sensitive (method).

Appendix C.5. Since draft-ietf-httpbis-p2-semantics-03

Closed issues:

*http://tools.ietf.org/wg/httpbis/trac/ticket/98: "OPTIONS request bodies"

*http://tools.ietf.org/wg/httpbis/trac/ticket/119: "Description of CONNECT should refer to RFC2817"

*http://tools.ietf.org/wg/httpbis/trac/ticket/125: "Location Content-Location reference request/response mixup"

Ongoing work on Method Registry (http://tools.ietf.org/wg/httpbis/trac/ticket/72):

*Added initial proposal for registration process, plus initial content (non-HTTP/1.1 methods to be added by a separate specification).

Appendix C.6. Since draft-ietf-httpbis-p2-semantics-04

Closed issues:

http://tools.ietf.org/wg/httpbis/trac/ticket/103: "Content-"

*http://tools.ietf.org/wg/httpbis/trac/ticket/132: "RFC 2822 is updated by RFC 5322"

Ongoing work on ABNF conversion (http://tools.ietf.org/wg/httpbis/trac/ticket/36):

*Use "/" instead of "|" for alternatives.

*Introduce new ABNF rules for "bad" whitespace ("BWS"), optional whitespace ("OWS") and required whitespace ("RWS").

* Rewrite ABNFs to spell out whitespace rules, factor out header field value format definitions.

Appendix C.7. Since draft-ietf-httpbis-p2-semantics-05

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/94: "Reason-Phrase BNF"

Final work on ABNF conversion (http://tools.ietf.org/wg/httpbis/trac/ticket/36):

* Add appendix containing collected and expanded ABNF, reorganize ABNF introduction.

Appendix C.8. Since draft-ietf-httpbis-p2-semantics-06

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/144: "Clarify when Referer is sent"

* http://tools.ietf.org/wg/httpbis/trac/ticket/164: "status codes vs methods"

* http://tools.ietf.org/wg/httpbis/trac/ticket/170: "Do not require "updates" relation for specs that register status codes or method names"

Appendix C.9. Since draft-ietf-httpbis-p2-semantics-07

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/27: "Idempotency"

* http://tools.ietf.org/wg/httpbis/trac/ticket/33: "TRACE security considerations"

* http://tools.ietf.org/wg/httpbis/trac/ticket/110: "Clarify rules for determining what entities a response carries"

* http://tools.ietf.org/wg/httpbis/trac/ticket/140: "update note citing RFC 1945 and 2068"

* http://tools.ietf.org/wg/httpbis/trac/ticket/182: "update note about redirect limit"

* http://tools.ietf.org/wg/httpbis/trac/ticket/191: "Location header ABNF should use 'URI'"

* http://tools.ietf.org/wg/httpbis/trac/ticket/192: "fragments in Location vs status 303"

* http://tools.ietf.org/wg/httpbis/trac/ticket/198: "move IANA registrations for optional status codes"

Partly resolved issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/171: "Are OPTIONS and TRACE safe?"

Appendix C.10. Since draft-ietf-httpbis-p2-semantics-08

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/10: "Safe Methods vs Redirection" (we missed the introduction to the 3xx status codes when fixing this previously)

Appendix C.11. Since draft-ietf-httpbis-p2-semantics-09

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/43: "Fragment combination / precedence during redirects"

Partly resolved issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/185: "Location header payload handling"

* http://tools.ietf.org/wg/httpbis/trac/ticket/196: "Term for the requested resource's URI"

Appendix C.12. Since draft-ietf-httpbis-p2-semantics-10

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/69: "Clarify 'Requested Variant'"

* http://tools.ietf.org/wg/httpbis/trac/ticket/109: "Clarify entity / representation / variant terminology"

* http://tools.ietf.org/wg/httpbis/trac/ticket/139: "Methods and Caching"

* http://tools.ietf.org/wg/httpbis/trac/ticket/190: "OPTIONS vs Max-Forwards"

* http://tools.ietf.org/wg/httpbis/trac/ticket/199: "Status codes and caching"

* http://tools.ietf.org/wg/httpbis/trac/ticket/220: "consider removing the 'changes from 2068' sections"

Appendix C.13. Since draft-ietf-httpbis-p2-semantics-11

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/229: "Considerations for new status codes"

* http://tools.ietf.org/wg/httpbis/trac/ticket/230: "Considerations for new methods"

* http://tools.ietf.org/wg/httpbis/trac/ticket/232: "User-Agent guidelines" (relating to the 'User-Agent' header field)

Appendix C.14. Since draft-ietf-httpbis-p2-semantics-12

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/43: "Fragment combination / precedence during redirects" (added warning about having a fragid on the redirect may cause inconvenience in some cases)

* http://tools.ietf.org/wg/httpbis/trac/ticket/79: "Content-* vs. PUT"

* http://tools.ietf.org/wg/httpbis/trac/ticket/88: "205 Bodies"

* http://tools.ietf.org/wg/httpbis/trac/ticket/102: "Understanding Content-* on non-PUT requests"

* http://tools.ietf.org/wg/httpbis/trac/ticket/103: "Content-*"

* http://tools.ietf.org/wg/httpbis/trac/ticket/104: "Header type defaulting"

* http://tools.ietf.org/wg/httpbis/trac/ticket/112: "PUT - 'store under' vs 'store at'"

* http://tools.ietf.org/wg/httpbis/trac/ticket/137: "duplicate ABNF for Reason-Phrase"

* http://tools.ietf.org/wg/httpbis/trac/ticket/180: "Note special status of Content-* prefix in header registration procedures"

* http://tools.ietf.org/wg/httpbis/trac/ticket/203: "Max-Forwards vs extension methods"

* http://tools.ietf.org/wg/httpbis/trac/ticket/213: "What is the value space of HTTP status codes?" (actually fixed in draft-ietf-httpbis-p2-semantics-11)

* http://tools.ietf.org/wg/httpbis/trac/ticket/224: "Header Classification"

* http://tools.ietf.org/wg/httpbis/trac/ticket/225: "PUT side effect: invalidation or just stale?"

* http://tools.ietf.org/wg/httpbis/trac/ticket/226: "proxies not supporting certain methods"

* http://tools.ietf.org/wg/httpbis/trac/ticket/239: "Migrate CONNECT from RFC2817 to p2"

* http://tools.ietf.org/wg/httpbis/trac/ticket/240: "Migrate Upgrade details from RFC2817"

* http://tools.ietf.org/wg/httpbis/trac/ticket/267: "clarify PUT semantics'"

* http://tools.ietf.org/wg/httpbis/trac/ticket/275: "duplicate ABNF for 'Method'"

* http://tools.ietf.org/wg/httpbis/trac/ticket/276: "untangle ABNFs for header fields"

Appendix C.15. Since draft-ietf-httpbis-p2-semantics-13

Closed issues:

* http://tools.ietf.org/wg/httpbis/trac/ticket/276: "untangle ABNFs for header fields"

* http://tools.ietf.org/wg/httpbis/trac/ticket/251: "message-body in CONNECT request"

Authors' Addresses

Roy T. Fielding editor Fielding Adobe Systems Incorporated 345 Park Ave San Jose, CA 95110 USA EMail: fielding@gbiv.com URI: http://roy.gbiv.com/

Jim Gettys Gettys Alcatel-Lucent Bell Labs 21 Oak Knoll Road Carlisle, MA 01741 USA EMail: jg@freedesktop.org URI: http://gettys.wordpress.com/

Jeffrey C. Mogul Mogul Hewlett-Packard Company HP Labs, Large Scale Systems Group 1501 Page Mill Road, MS 1177 Palo Alto, CA 94304 USA EMail: JeffMogul@acm.org

Henrik Frystyk Nielsen Frystyk Microsoft Corporation
1 Microsoft Way Redmond, WA 98052 USA EMail: henrikn@microsoft.com

Larry Masinter Masinter Adobe Systems Incorporated 345 Park Ave San Jose, CA 95110 USA EMail: LMM@acm.org URI: http://larry.masinter.net/

Paul J. Leach Leach Microsoft Corporation 1 Microsoft Way Redmond, WA 98052 EMail: paulle@microsoft.com

Tim Berners-Lee Berners-Lee World Wide Web Consortium MIT Computer Science and Artificial Intelligence Laboratory The Stata Center, Building 32 32 Vassar Street Cambridge, MA 02139 USA EMail: timbl@w3.org URI: http://www.w3.org/People/Berners-Lee/

Yves Lafon editor Lafon World Wide Web Consortium W3C / ERCIM 2004, rte des Lucioles Sophia-Antipolis, AM 06902 France EMail: ylafon@w3.org URI: http://www.raubacapeu.net/people/yves/

Julian F. Reschke editor Reschke greenbytes GmbH Hafenweg 16 Muenster, NW 48155 Germany Phone: +49 251 2807760 EMail: julian.reschke@greenbytes.de URI: http://greenbytes.de/tech/webdav/