HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Updates: 2817 (if approved)
Intended status: Standards Track
Expires: January 17, 2013

R. Fielding, Ed.
Adobe
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes
July 16, 2012

HTTP/1.1, part 2: Semantics and Payloads
draft-ietf-httpbis-p2-semantics-20

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypertext information systems. This document defines the semantics of HTTP/1.1 messages, as expressed by request methods, request header fields, response status codes, and response header fields, along with the payload of messages (metadata and body content) and mechanisms for content negotiation.

Editorial Note (To be removed by RFC Editor)

Discussion of this draft takes 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 F.40.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference

material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 17, 2013.

Table of Contents

1. Introduction

Each HTTP message is either a request or a response. A server listens on a connection for a request, parses each message received, interprets the message semantics in relation to the identified request target, and responds to that request with one or more response messages. This document defines HTTP/1.1 request and response semantics in terms of the architecture, syntax notation, and conformance criteria defined in [Part1].

HTTP provides a uniform interface for interacting with resources regardless of their type, nature, or implementation. HTTP semantics includes the intentions defined by each request method, extensions to those semantics that might be described in request header fields, the meaning of status codes to indicate a machine-readable response, and additional control data and resource metadata that might be given in response header fields.

In addition, this document defines the payload of messages (a.k.a., content), the associated metadata header fields that define how the payload is intended to be interpreted by a recipient, the request header fields that might influence content selection, and the various selection algorithms that are collectively referred to as "content negotiation".

> Note: This document is currently disorganized in order to minimize 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. Conformance and Error Handling

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].

This specification targets conformance criteria according to the role of a participant in HTTP communication. Hence, HTTP requirements are placed on senders, recipients, clients, servers, user agents, intermediaries, origin servers, proxies, gateways, or caches, depending on what behavior is being constrained by the requirement. See Section 2 of [Part1] for definitions of these terms.

The verb "generate" is used instead of "send" where a requirement differentiates between creating a protocol element and merely forwarding a received element downstream.

An implementation is considered conformant if it complies with all of the requirements associated with the roles it partakes in HTTP. Note that SHOULD-level requirements are relevant here, unless one of the documented exceptions is applicable.

This document also uses ABNF to define valid protocol elements (Section 1.2). In addition to the prose requirements placed upon them, senders MUST NOT generate protocol elements that do not match the grammar defined by the ABNF rules for those protocol elements that are applicable to the sender's role. If a received protocol element is processed, the recipient MUST be able to parse any value that would match the ABNF rules for that protocol element, excluding only those rules not applicable to the recipient's role.

Unless noted otherwise, a recipient MAY attempt to recover a usable protocol element from an invalid construct. HTTP does not define specific error handling mechanisms except when they have a direct impact on security, since different applications of the protocol require different error handling strategies. For example, a Web browser might wish to transparently recover from a response where the Location header field doesn't parse according to the ABNF, whereas a systems control client might consider any form of error recovery to be dangerous.

1.2. Syntax Notation

This specification uses the Augmented Backus-Naur Form (ABNF) notation of [RFC5234] with the list rule extension defined in Section 1.2 of [Part1]. Appendix D describes rules imported from other documents. Appendix E shows the collected ABNF with the list rule expanded.

2. Methods

The method token indicates the request method to be performed on the target resource (Section 5.5 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 9.5). 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 2.3.

2.1. Safe and Idempotent Methods

2.1.1. Safe Methods

Implementers 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.

2.1.2. 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.

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 2.1.1)
- Idempotent ("yes" or "no", see Section 2.1.1)
- Pointer to specification text

Values to be added to this name space require IETF Review (see [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 might 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 2.1.1), what semantics (if any) the request body has, and whether they are idempotent (Section 2.1.2). They also need to state whether they can be cached ([Part6]); in particular under what conditions a cache can store the response, and under what conditions such a stored response can be used to satisfy a subsequent request.

2.3. Method Definitions

2.3.1. 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 (Section 5.3 of [Part1]) 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 conformance (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 (OK) 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 9.14). If no Max-Forwards field is present in the request, then the forwarded request MUST NOT include a Max-Forwards field.

2.3.2. 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 ([Part4]). 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 ([Part5]). 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.

2.3.3. 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. It also has potential side effects on previously stored responses to GET; see Section 5 of [Part6].

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.

2.3.4. 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 9.13).

Responses to POST requests are only cacheable when they include explicit freshness information (see Section 4.1.1 of [Part6]). A cached POST response with a Content-Location header field (see Section 9.8) 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 representation of the resource.

2.3.5. 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 (Unsupported Media Type) 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 (Section 5.2 of [Part5]), 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 6 of [Part6]).

2.3.6. 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 a 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 6 of [Part6]).

2.3.7. 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 9.14). 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 6.2 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 7.3.1 of [Part1]) and contain a message body that encloses a copy of the entire request message. Responses to the TRACE method are not cacheable.

2.3.8. CONNECT

The CONNECT method requests that the proxy establish a tunnel to the request-target and, if successful, thereafter 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 5.3 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
```

Any 2xx (Successful) 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. The tunneled data from the server begins immediately after the blank line that concludes the successful response’s header block.

A server SHOULD NOT send any Transfer-Encoding or Content-Length header fields in a successful response. A client MUST ignore any Content-Length or Transfer-Encoding header fields received in a successful response.

Any response other than a successful response indicates that the tunnel has not yet been formed and that the connection remains governed by HTTP.

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=
```

A message body on a CONNECT request has no defined semantics. Sending a body on a CONNECT request might cause existing implementations to reject the request.

Similar to a pipelined HTTP/1.1 request, data to be tunneled from client to server MAY be sent immediately after the request (before a response is received). The usual caveats also apply: data can be discarded if the eventual response is negative, and the connection can be reset with no response if more than one TCP segment is outstanding.

It might 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.

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.

An origin server which receives a CONNECT request for itself MAY respond with a 2xx status code to indicate that a connection is established. However, most origin servers do not implement CONNECT.

3. Header Fields

Header fields are key value pairs that can be used to communicate data about the message, its payload, the target resource, or about the connection itself (i.e., control data). See Section 3.2 of [Part1] for a general definition of their syntax.

3.1. Considerations for Creating Header Fields

New header fields are registered using the procedures described in [RFC3864].

The requirements for header field names are defined in Section 4.1 of [RFC3864]. Authors of specifications defining new fields are advised to keep the name as short as practical, and not to prefix them with "X-" if they are to be registered (either immediately or in the future).

New header field values typically have their syntax defined using ABNF ([RFC5234]), using the extension defined in Appendix B of [Part1] as necessary, and are usually constrained to the range of ASCII characters. Header fields needing a greater range of characters can use an encoding such as the one defined in [RFC5987].

Because commas (",") are used as a generic delimiter between field-values, they need to be treated with care if they are allowed in the field-value's payload. Typically, components that might contain a comma are protected with double-quotes using the quoted-string ABNF production (Section 3.2.4 of [Part1]).

For example, a textual date and a URI (either of which might contain a comma) could be safely carried in field-values like these:

```
  Example-URI-Field: "http://example.com/a.html,foo",
                     "http://without-a-comma.example.com/"
  Example-Date-Field: "Sat, 04 May 1996", "Wed, 14 Sep 2005"
```

Note that double quote delimiters almost always are used with the quoted-string production; using a different syntax inside double quotes will likely cause unnecessary confusion.

Many header fields use a format including (case-insensitively) named parameters (for instance, Content-Type, defined in Section 9.9). Allowing both unquoted (token) and quoted (quoted-string) syntax for the parameter value enables recipients to use existing parser components. When allowing both forms, the meaning of a parameter value ought to be independent of the syntax used for it (for an example, see the notes on parameter handling for media types in Section 5.5).

Authors of specifications defining new header fields are advised to consider documenting:

o Whether the field is a single value, or whether it can be a list (delimited by commas; see Section 3.2 of [Part1]).

  If it does not use the list syntax, document how to treat messages where the header field occurs multiple times (a sensible default would be to ignore the header field, but this might not always be the right choice).

Note that intermediaries and software libraries might combine multiple header field instances into a single one, despite the header field not allowing this. A robust format enables recipients to discover these situations (good example: "Content-Type", as the comma can only appear inside quoted strings; bad example: "Location", as a comma can occur inside a URI).

- Under what conditions the header field can be used; e.g., only in responses or requests, in all messages, only on responses to a particular request method.
- Whether it is appropriate to list the field-name in the Connection header field (i.e., if the header field is to be hop-by-hop, see Section 6.1 of [Part1]).
- Under what conditions intermediaries are allowed to modify the header field's value, insert or delete it.
- How the header field might interact with caching (see [Part6]).
- Whether the header field is useful or allowable in trailers (see Section 4.1 of [Part1]).
- Whether the header field ought to be preserved across redirects.

3.2. 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 9.1
Accept-Charset	Section 9.2
Accept-Encoding	Section 9.3
Accept-Language	Section 9.4
Authorization	Section 4.1 of [Part7]
Expect	Section 9.11
From	Section 9.12
Host	Section 5.4 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 9.14
Proxy-Authorization	Section 4.3 of [Part7]
Range	Section 5.4 of [Part5]
Referer	Section 9.15
TE	Section 4.3 of [Part1]
User-Agent	Section 9.18

3.3. 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 5.5 of [Part1]).

Header Field Name	Defined in...
Accept-Ranges	Section 5.1 of [Part5]
Age	Section 7.1 of [Part6]
Allow	Section 9.5
Date	Section 9.10
ETag	Section 2.3 of [Part4]
Location	Section 9.13
Proxy-Authenticate	Section 4.2 of [Part7]
Retry-After	Section 9.16
Server	Section 9.17
Vary	Section 7.5 of [Part6]
WWW-Authenticate	Section 4.4 of [Part7]

4. Status Codes

The status-code element is a 3-digit integer result code of the attempt to understand and satisfy the request.

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.

The first digit of the status-code defines the class of response. The last two digits do not have any categorization role. There are 5 values for the first digit:

- 1xx (Informational): Request received, continuing process
- 2xx (Successful): The action was successfully received, understood, and accepted
- 3xx (Redirection): Further action needs to be taken in order to complete the request
- 4xx (Client Error): The request contains bad syntax or cannot be fulfilled
- 5xx (Server Error): The server failed to fulfill an apparently valid request

For most status codes the response can carry a payload, in which case a Content-Type header field indicates the payload’s media type (Section 9.9).

4.1. Overview of Status Codes

The status codes listed below are defined in 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 4.3.1
101	Switching Protocols	Section 4.3.2
200	OK	Section 4.4.1
201	Created	Section 4.4.2
202	Accepted	Section 4.4.3
203	Non-Authoritative Information	Section 4.4.4
204	No Content	Section 4.4.5
205	Reset Content	Section 4.4.6
206	Partial Content	Section 3.1 of [Part5]
300	Multiple Choices	Section 4.5.1
301	Moved Permanently	Section 4.5.2
302	Found	Section 4.5.3
303	See Other	Section 4.5.4
304	Not Modified	Section 4.1 of [Part4]
305	Use Proxy	Section 4.5.5
307	Temporary Redirect	Section 4.5.7
400	Bad Request	Section 4.6.1
401	Unauthorized	Section 3.1 of [Part7]
402	Payment Required	Section 4.6.2
403	Forbidden	Section 4.6.3
404	Not Found	Section 4.6.4
405	Method Not Allowed	Section 4.6.5
406	Not Acceptable	Section 4.6.6
407	Proxy Authentication Required	Section 3.2 of [Part7]
408	Request Time-out	Section 4.6.7
409	Conflict	Section 4.6.8
410	Gone	Section 4.6.9
411	Length Required	Section 4.6.10
412	Precondition Failed	Section 4.2 of [Part4]
413	Request Representation Too Large	Section 4.6.11
414	URI Too Long	Section 4.6.12
415	Unsupported Media Type	Section 4.6.13
416	Requested range not satisfiable	Section 3.2 of [Part5]
417	Expectation Failed	Section 4.6.14
426	Upgrade Required	Section 4.6.15
500	Internal Server Error	Section 4.7.1
501	Not Implemented	Section 4.7.2

502	Bad Gateway	Section 4.7.3
503	Service Unavailable	Section 4.7.4
504	Gateway Time-out	Section 4.7.5
505	HTTP Version not supported	Section 4.7.6

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 require IETF Review (see [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 header fields and/or method(s)), along with any interactions with response header fields (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 4. 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 field(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 7.1; by default, it is anonymous).

4.3. 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 an "Expect: 100-continue" field when it forwards a request, then it need not forward the corresponding 100 (Continue) response(s).)

4.3.1. 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 6.4.3 of [Part1] for detailed discussion of the use and handling of this status code.

4.3.2. 101 Switching Protocols

The server understands and is willing to comply with the client’s request, via the Upgrade message header field (Section 6.5 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.

4.4. Successful 2xx

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

4.4.1. 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 4.1.2 of [Part6]) to determine freshness for 200 responses.

4.4.2. 201 Created

The request has been fulfilled and has resulted in one or more new resources being created.

Newly created resources are typically linked to from the response payload, with the most relevant URI also being carried in the Location header field. If the newly created resource's URI is the same as the Effective Request URI, this information can be omitted (e.g., in the case of a response to a PUT request).

The origin server MUST create the resource(s) 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 identified by the Location header field or, in case the Location header field was omitted, by the Effective Request URI (see Section 2.3 of [Part4]).

4.4.3. 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.

4.4.4. 203 Non-Authoritative Information

The representation in the response has been transformed or otherwise modified by a transforming proxy (Section 2.4 of [Part1]). Note that the behavior of transforming intermediaries is controlled by the no-transform Cache-Control directive (Section 7.2 of [Part6]).

This status code is only appropriate when the response status code would have been 200 (OK) otherwise. When the status code before transformation would have been different, the 214 Transformation Applied warn-code (Section 7.6 of [Part6]) is appropriate.

Caches MAY use a heuristic (see Section 4.1.2 of [Part6]) to determine freshness for 203 responses.

4.4.5. 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.

4.4.6. 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].

4.5. 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. If the required action involves a subsequent HTTP request, it 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 2.1.1.

There are several types of redirects:

1. Redirects of the request to another URI, either temporarily or permanently. The new URI is specified in the Location header field. In this specification, the status codes 301 (Moved Permanently), 302 (Found), and 307 (Temporary Redirect) fall under this category.

2. Redirection to a new location that represents an indirect response to the request, such as the result of a POST operation to be retrieved with a subsequent GET request. This is status code 303 (See Other).

3. Redirection offering a choice of matching resources for use by agent-driven content negotiation (Section 8.2). This is status code 300 (Multiple Choices).

4. Other kinds of redirection, such as to a cached result (status code 304 (Not Modified), see Section 4.1 of [Part4]).

Note: In HTTP/1.0, only the status codes 301 (Moved Permanently) and 302 (Found) were defined for the first type of redirect, and the second type did not exist at all ([RFC1945], Section 9.3). However it turned out that web forms using POST expected redirects to change the operation for the subsequent request to retrieval (GET). To address this use case, HTTP/1.1 introduced the second type of redirect with the status code 303 (See Other) ([RFC2068], Section 10.3.4). As user agents did not change their behavior to maintain backwards compatibility, the first revision of HTTP/1.1 added yet another status code, 307 (Temporary Redirect), for which the backwards compatibility problems did not apply ([RFC2616], Section 10.3.8). Over 10 years later, most user agents still do method rewriting for status codes 301 and 302, therefore this specification makes that behavior conformant in case the original request was POST.

A Location header field on a 3xx response indicates that a client MAY automatically redirect to the URI provided; see Section 9.13.

Note that for methods not known to be "safe", as defined in Section 2.1.1, automatic redirection needs to done with care, since the redirect might change the conditions under which the request was issued.

Clients SHOULD detect and intervene in cyclical redirections (i.e., "infinite" redirection loops).

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.

4.5.1. 300 Multiple Choices

The target resource has more than one representation, each with its own specific location, and agent-driven negotiation information (Section 8) 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. 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 4.1.2 of [Part6]) to determine freshness for 300 responses.

4.5.2. 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 4.1.2 of [Part6]) to determine freshness for 301 responses.

The new permanent URI SHOULD be given by the Location field in the response. A response payload can contain a short hypertext note with a hyperlink to the new URI(s).

> Note: For historic reasons, user agents MAY change the request method from POST to GET for the subsequent request. If this behavior is undesired, status code 307 (Temporary Redirect) can be used instead.

4.5.3. 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. A response payload can contain a short hypertext note with a hyperlink to the new URI(s).

> Note: For historic reasons, user agents MAY change the request method from POST to GET for the subsequent request. If this behavior is undesired, status code 307 (Temporary Redirect) can be

used instead.

4.5.4. 303 See Other

The 303 status code indicates that the server is redirecting the user agent to a different resource, as indicated by a URI in the Location header field, that is intended to provide an indirect response to the original request. In order to satisfy the original request, a user agent SHOULD perform a retrieval request using the Location URI (a GET or HEAD request if using HTTP), which can itself be redirected further, and present the eventual result as an answer to the original request. Note that the new URI in the Location header field is not considered equivalent to the effective request URI.

This 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.

4.5.5. 305 Use Proxy

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

4.5.6. 306 (Unused)

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

4.5.7. 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. A response payload can contain a short hypertext note with a hyperlink to the new URI(s).

> Note: This status code is similar to 302 (Found), except that it does not allow rewriting the request method from POST to GET. This specification defines no equivalent counterpart for 301 (Moved Permanently) ([draft-reschke-http-status-308], however, defines the status code 308 (Permanent Redirect) for this purpose).

4.6. 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.

4.6.1. 400 Bad Request

The server cannot or will not process the request, due to a client error (e.g., malformed syntax).

4.6.2. 402 Payment Required

This code is reserved for future use.

4.6.3. 403 Forbidden

The server understood the request, but refuses to authorize it. Providing different user authentication credentials might be successful, but any credentials that were provided in the request are insufficient. The request SHOULD NOT be repeated with the same credentials.

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) MAY be used instead.

4.6.4. 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.

4.6.5. 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.

4.6.6. 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 and 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. 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.

4.6.7. 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.

4.6.8. 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.

4.6.9. 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 4.1.2 of [Part6]) to determine freshness for 410 responses.

4.6.10. 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.

4.6.11. 413 Request Representation 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.

4.6.12. 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 request-target.

4.6.13. 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.

4.6.14. 417 Expectation Failed

The expectation given in an Expect header field (see Section 9.11) 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.

4.6.15. 426 Upgrade Required

The request can not be completed without a prior protocol upgrade. This response MUST include an Upgrade header field (Section 6.5 of [Part1]) specifying the required protocols.

Example:

```
HTTP/1.1 426 Upgrade Required
Upgrade: HTTP/3.0
Connection: Upgrade
Content-Length: 53
Content-Type: text/plain

This service requires use of the HTTP/3.0 protocol.
```

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 might be available to the user.

4.7. 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.

4.7.1. 500 Internal Server Error

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

4.7.2. 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.

4.7.3. 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.

4.7.4. 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 (Section 9.16). If no Retry-After is given, the client SHOULD handle the response as it would for a 500 (Internal Server Error) response.

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

4.7.5. 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 implementers: some deployed proxies are known to return 400 (Bad Request) or 500 (Internal Server Error) when DNS lookups time out.

4.7.6. 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 2.7 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.

5. Protocol Parameters

5.1. Date/Time Formats

HTTP applications have historically allowed three different formats for date/time stamps. However, the preferred format is a fixed-length subset of that defined by [RFC1123]:

```
Sun, 06 Nov 1994 08:49:37 GMT  ; RFC 1123
```

The other formats are described here only for compatibility with obsolete implementations.

```
Sunday, 06-Nov-94 08:49:37 GMT ; obsolete RFC 850 format
Sun Nov  6 08:49:37 1994       ; ANSI C's asctime() format
```

HTTP/1.1 clients and servers that parse a date value MUST accept all

three formats (for compatibility with HTTP/1.0), though they MUST only generate the RFC 1123 format for representing HTTP-date values in header fields.

All HTTP date/time stamps MUST be represented in Greenwich Mean Time (GMT), without exception. For the purposes of HTTP, GMT is exactly equal to UTC (Coordinated Universal Time). This is indicated in the first two formats by the inclusion of "GMT" as the three-letter abbreviation for time zone, and MUST be assumed when reading the asctime format. HTTP-date is case sensitive and MUST NOT include additional whitespace beyond that specifically included as SP in the grammar.

```
  HTTP-date    = rfc1123-date / obs-date
```

Preferred format:

```
rfc1123-date = day-name "," SP date1 SP time-of-day SP GMT
; fixed length subset of the format defined in
; Section 5.2.14 of [RFC1123]

day-name     = %x4D.6F.6E ; "Mon", case-sensitive
             / %x54.75.65 ; "Tue", case-sensitive
             / %x57.65.64 ; "Wed", case-sensitive
             / %x54.68.75 ; "Thu", case-sensitive
             / %x46.72.69 ; "Fri", case-sensitive
             / %x53.61.74 ; "Sat", case-sensitive
             / %x53.75.6E ; "Sun", case-sensitive

date1        = day SP month SP year
             ; e.g., 02 Jun 1982

day          = 2DIGIT
month        = %x4A.61.6E ; "Jan", case-sensitive
             / %x46.65.62 ; "Feb", case-sensitive
             / %x4D.61.72 ; "Mar", case-sensitive
             / %x41.70.72 ; "Apr", case-sensitive
             / %x4D.61.79 ; "May", case-sensitive
             / %x4A.75.6E ; "Jun", case-sensitive
             / %x4A.75.6C ; "Jul", case-sensitive
             / %x41.75.67 ; "Aug", case-sensitive
             / %x53.65.70 ; "Sep", case-sensitive
             / %x4F.63.74 ; "Oct", case-sensitive
             / %x4E.6F.76 ; "Nov", case-sensitive
             / %x44.65.63 ; "Dec", case-sensitive
year         = 4DIGIT

GMT   = %x47.4D.54 ; "GMT", case-sensitive

time-of-day  = hour ":" minute ":" second
               ; 00:00:00 - 23:59:59

hour         = 2DIGIT
minute       = 2DIGIT
second       = 2DIGIT
```

The semantics of day-name, day, month, year, and time-of-day are the same as those defined for the RFC 5322 constructs with the corresponding name ([RFC5322], Section 3.3).

Obsolete formats:

```
obs-date     = rfc850-date / asctime-date
```

```
rfc850-date  = day-name-l "," SP date2 SP time-of-day SP GMT
date2        = day "-" month "-" 2DIGIT
               ; day-month-year (e.g., 02-Jun-82)

day-name-l   = %x4D.6F.6E.64.61.79 ; "Monday", case-sensitive
       / %x54.75.65.73.64.61.79 ; "Tuesday", case-sensitive
       / %x57.65.64.6E.65.73.64.61.79 ; "Wednesday", case-sensitive
       / %x54.68.75.72.73.64.61.79 ; "Thursday", case-sensitive
       / %x46.72.69.64.61.79 ; "Friday", case-sensitive
       / %x53.61.74.75.72.64.61.79 ; "Saturday", case-sensitive
       / %x53.75.6E.64.61.79 ; "Sunday", case-sensitive


asctime-date = day-name SP date3 SP time-of-day SP year
date3        = month SP ( 2DIGIT / ( SP 1DIGIT ))
               ; month day (e.g., Jun  2)
```

Note: Recipients of date values are encouraged to be robust in accepting date values that might have been sent by non-HTTP applications, as is sometimes the case when retrieving or posting messages via proxies/gateways to SMTP or NNTP.

Note: HTTP requirements for the date/time stamp format apply only to their usage within the protocol stream. Clients and servers are not required to use these formats for user presentation, request logging, etc.

5.2. Product Tokens

Product tokens are used to allow communicating applications to identify themselves by software name and version. Most fields using product tokens also allow sub-products which form a significant part of the application to be listed, separated by whitespace. By convention, the products are listed in order of their significance for identifying the application.

```
product         = token ["/" product-version]
product-version = token
```

Examples:

```
User-Agent: CERN-LineMode/2.15 libwww/2.17b3
Server: Apache/0.8.4
```

Product tokens SHOULD be short and to the point. They MUST NOT be used for advertising or other non-essential information. Although any token octet MAY appear in a product-version, this token SHOULD only be used for a version identifier (i.e., successive versions of

the same product SHOULD only differ in the product-version portion of the product value).

5.3. Character Encodings (charset)

HTTP uses charset names to indicate the character encoding of a textual representation.

A character encoding is identified by a case-insensitive token. The complete set of tokens is defined by the IANA Character Set registry (<http://www.iana.org/assignments/character-sets>).

```
charset = token
```

Although HTTP allows an arbitrary token to be used as a charset value, any token that has a predefined value within the IANA Character Set registry MUST represent the character encoding defined by that registry. Applications SHOULD limit their use of character encodings to those defined within the IANA registry.

HTTP uses charset in two contexts: within an Accept-Charset request header field (in which the charset value is an unquoted token) and as the value of a parameter in a Content-Type header field (within a request or response), in which case the parameter value of the charset parameter can be quoted.

Implementers need to be aware of IETF character set requirements [RFC3629] [RFC2277].

5.4. Content Codings

Content coding values indicate an encoding transformation that has been or can be applied to a representation. Content codings are primarily used to allow a representation to be compressed or otherwise usefully transformed without losing the identity of its underlying media type and without loss of information. Frequently, the representation is stored in coded form, transmitted directly, and only decoded by the recipient.

```
content-coding   = token
```

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding (Section 9.3) and Content-Encoding (Section 9.6) header fields. Although the value describes the content-coding, what is more important is that it indicates what decoding mechanism will be required to remove the encoding.

compress

See Section 4.2.1 of [Part1].

deflate

See Section 4.2.2 of [Part1].

gzip

See Section 4.2.3 of [Part1].

5.4.1. Content Coding Registry

The HTTP Content Coding Registry defines the name space for the content coding names.

Registrations MUST include the following fields:

- Name
- Description
- Pointer to specification text

Names of content codings MUST NOT overlap with names of transfer codings (Section 4 of [Part1]), unless the encoding transformation is identical (as is the case for the compression codings defined in Section 4.2 of [Part1]).

Values to be added to this name space require IETF Review (see Section 4.1 of [RFC5226]), and MUST conform to the purpose of content coding defined in this section.

The registry itself is maintained at <http://www.iana.org/assignments/http-parameters>.

5.5. Media Types

HTTP uses Internet Media Types [RFC2046] in the Content-Type (Section 9.9) and Accept (Section 9.1) header fields in order to provide open and extensible data typing and type negotiation.

```
  media-type = type "/" subtype *( OWS ";" OWS parameter )
  type       = token
  subtype    = token
```

The type/subtype MAY be followed by parameters in the form of

attribute/value pairs.

```
  parameter      = attribute "=" value
  attribute      = token
  value          = word
```

The type, subtype, and parameter attribute names are case-insensitive. Parameter values might or might not be case-sensitive, depending on the semantics of the parameter name. The presence or absence of a parameter might be significant to the processing of a media-type, depending on its definition within the media type registry.

A parameter value that matches the token production can be transmitted as either a token or within a quoted-string. The quoted and unquoted values are equivalent.

Note that some older HTTP applications do not recognize media type parameters. When sending data to older HTTP applications, implementations SHOULD only use media type parameters when they are required by that type/subtype definition.

Media-type values are registered with the Internet Assigned Number Authority (IANA). The media type registration process is outlined in [RFC4288]. Use of non-registered media types is discouraged.

5.5.1. Canonicalization and Text Defaults

Internet media types are registered with a canonical form. A representation transferred via HTTP messages MUST be in the appropriate canonical form prior to its transmission except for "text" types, as defined in the next paragraph.

When in canonical form, media subtypes of the "text" type use CRLF as the text line break. HTTP relaxes this requirement and allows the transport of text media with plain CR or LF alone representing a line break when it is done consistently for an entire representation. HTTP applications MUST accept CRLF, bare CR, and bare LF as indicating a line break in text media received via HTTP. In addition, if the text is in a character encoding that does not use octets 13 and 10 for CR and LF respectively, as is the case for some multi-byte character encodings, HTTP allows the use of whatever octet sequences are defined by that character encoding to represent the equivalent of CR and LF for line breaks. This flexibility regarding line breaks applies only to text media in the payload body; a bare CR or LF MUST NOT be substituted for CRLF within any of the HTTP control structures (such as header fields and multipart boundaries).

If a representation is encoded with a content-coding, the underlying data MUST be in a form defined above prior to being encoded.

5.5.2. Multipart Types

MIME provides for a number of "multipart" types -- encapsulations of one or more representations within a single message body. All multipart types share a common syntax, as defined in Section 5.1.1 of [RFC2046], and MUST include a boundary parameter as part of the media type value. The message body is itself a protocol element and MUST therefore use only CRLF to represent line breaks between body-parts.

In general, HTTP treats a multipart message body no differently than any other media type: strictly as payload. HTTP does not use the multipart boundary as an indicator of message body length. In all other respects, an HTTP user agent SHOULD follow the same or similar behavior as a MIME user agent would upon receipt of a multipart type. The MIME header fields within each body-part of a multipart message body do not have any significance to HTTP beyond that defined by their MIME semantics.

If an application receives an unrecognized multipart subtype, the application MUST treat it as being equivalent to "multipart/mixed".

> Note: The "multipart/form-data" type has been specifically defined for carrying form data suitable for processing via the POST request method, as described in [RFC2388].

5.6. Language Tags

A language tag, as defined in [RFC5646], identifies a natural language spoken, written, or otherwise conveyed by human beings for communication of information to other human beings. Computer languages are explicitly excluded. HTTP uses language tags within the Accept-Language and Content-Language fields.

In summary, a language tag is composed of one or more parts: A primary language subtag followed by a possibly empty series of subtags:

```
language-tag = <Language-Tag, defined in [RFC5646], Section 2.1>
```

White space is not allowed within the tag and all tags are case-insensitive. The name space of language subtags is administered by the IANA (see <http://www.iana.org/assignments/language-subtag-registry>).

Example tags include:

en, en-US, es-419, az-Arab, x-pig-latin, man-Nkoo-GN

See [RFC5646] for further information.

6. Payload

HTTP messages MAY transfer a payload if not otherwise restricted by the request method or response status code. The payload consists of metadata, in the form of header fields, and data, in the form of the sequence of octets in the message body after any transfer-coding has been decoded.

A "payload" in HTTP is always a partial or complete representation of some resource. We use separate terms for payload and representation because some messages contain only the associated representation's header fields (e.g., responses to HEAD) or only some part(s) of the representation (e.g., the 206 (Partial Content) status code).

6.1. Payload Header Fields

HTTP header fields that specifically define the payload, rather than the associated representation, are referred to as "payload header fields". The following payload header fields are defined by HTTP/1.1:

Header Field Name	Defined in...
Content-Length	Section 3.3.2 of [Part1]
Content-Range	Section 5.2 of [Part5]

6.2. Payload Body

A payload body is only present in a message when a message body is present, as described in Section 3.3 of [Part1]. The payload 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.

7. Representation

A "representation" is information in a format that can be readily communicated from one party to another. A resource representation is information that reflects the state of that resource, as observed at some point in the past (e.g., in a response to GET) or to be desired

at some point in the future (e.g., in a PUT request).

Most, but not all, representations transferred via HTTP are intended to be a representation of the target resource (the resource identified by the effective request URI). The precise semantics of a representation are determined by the type of message (request or response), the request method, the response status code, and the representation metadata. For example, the above semantic is true for the representation in any 200 (OK) response to GET and for the representation in any PUT request. A 200 response to PUT, in contrast, contains either a representation that describes the successful action or a representation of the target resource, with the latter indicated by a Content-Location header field with the same value as the effective request URI. Likewise, response messages with an error status code usually contain a representation that describes the error and what next steps are suggested for resolving it.

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.

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.

7.1. 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 5.5 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 (OK) or 203 (Non-Authoritative Information) and the request method was GET, the response payload is a representation of the target resource.

2. If the response status code is 204 (No Content), 206 (Partial Content), or 304 (Not Modified) and the request method was GET or HEAD, the response payload is a partial representation of the target resource.

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.

[[TODO-req-uri: The comparison function is going to have to be defined somewhere, because we already need to compare URIs for things like cache invalidation.]]

7.2. Representation Header Fields

Representation header fields define metadata about the representation data enclosed in the message body or, if no message body is present, about the representation that would have been transferred in a 200 (OK) response to a simultaneous GET request with the same effective request URI.

The following header fields are defined as representation metadata:

Header Field Name	Defined in...
Content-Encoding	Section 9.6
Content-Language	Section 9.7
Content-Location	Section 9.8
Content-Type	Section 9.9
Expires	Section 7.3 of [Part6]

We use the term "selected representation" to refer to the the current representation of a target resource that would have been selected in a successful response if the same request had used the method GET and excluded any conditional request header fields.

Additional header fields define metadata about the selected representation, which might differ from the representation included in the message for responses to some state-changing methods. The following header fields are defined as selected representation metadata:

Header Field Name	Defined in...
ETag	Section 2.3 of [Part4]
Last-Modified	Section 2.2 of [Part4]

7.3. Representation Data

The representation body associated with an HTTP message is either provided as the payload body of the message or referred to by the message semantics and the effective request URI. The representation data is in a format and encoding defined by the representation metadata header fields.

The data type of the representation data is determined via the header fields Content-Type and Content-Encoding. These define a two-layer, ordered encoding model:

```
representation-data := Content-Encoding( Content-Type( bits ) )
```

Content-Type specifies the media type of the underlying data, which defines both the data format and how that data SHOULD be processed by the recipient (within the scope of the request method semantics). Any HTTP/1.1 message containing a payload body SHOULD include a Content-Type header field defining the media type of the associated representation unless that metadata is unknown to the sender. If the Content-Type header field is not present, it indicates that the sender does not know the media type of the representation; recipients MAY either assume that the media type is "application/octet-stream" ([RFC2046], Section 4.5.1) or examine the content to determine its type.

In practice, resource owners do not always properly configure their origin server to provide the correct Content-Type for a given representation, with the result that some clients will examine a response body’s content and override the specified type. Clients that do so risk drawing incorrect conclusions, which might expose additional security risks (e.g., "privilege escalation"). Furthermore, it is impossible to determine the sender’s intent by examining the data format: many data formats match multiple media types that differ only in processing semantics. Implementers are

encouraged to provide a means of disabling such "content sniffing" when it is used.

Content-Encoding is used to indicate any additional content codings applied to the data, usually for the purpose of data compression, that are a property of the representation. If Content-Encoding is not present, then there is no additional encoding beyond that defined by the Content-Type header field.

8. Content Negotiation

HTTP responses include a representation which contains information for interpretation, whether by a human user or for further processing. Often, the server has different ways of representing the same information; for example, in different formats, languages, or using different character encodings.

HTTP clients and their users might have different or variable capabilities, characteristics or preferences which would influence which representation, among those available from the server, would be best for the server to deliver. For this reason, HTTP provides mechanisms for "content negotiation" -- a process of allowing selection of a representation of a given resource, when more than one is available.

This specification defines two patterns of content negotiation; "server-driven", where the server selects the representation based upon the client's stated preferences, and "agent-driven" negotiation, where the server provides a list of representations for the client to choose from, based upon their metadata. In addition, there are other patterns: some applications use an "active content" pattern, where the server returns active content which runs on the client and, based on client available parameters, selects additional resources to invoke. "Transparent Content Negotiation" ([RFC2295]) has also been proposed.

These patterns are all widely used, and have trade-offs in applicability and practicality. In particular, when the number of preferences or capabilities to be expressed by a client are large (such as when many different formats are supported by a user-agent), server-driven negotiation becomes unwieldy, and might not be appropriate. Conversely, when the number of representations to choose from is very large, agent-driven negotiation might not be appropriate.

Note that in all cases, the supplier of representations has the responsibility for determining which representations might be considered to be the "same information".

8.1. Server-driven Negotiation

If the selection of the best representation for a response is made by an algorithm located at the server, it is called server-driven negotiation. Selection is based on the available representations of the response (the dimensions over which it can vary; e.g., language, content-coding, etc.) and the contents of particular header fields in the request message or on other information pertaining to the request (such as the network address of the client).

Server-driven negotiation is advantageous when the algorithm for selecting from among the available representations is difficult to describe to the user agent, or when the server desires to send its "best guess" to the client along with the first response (hoping to avoid the round-trip delay of a subsequent request if the "best guess" is good enough for the user). In order to improve the server's guess, the user agent MAY include request header fields (Accept, Accept-Language, Accept-Encoding, etc.) which describe its preferences for such a response.

Server-driven negotiation has disadvantages:

1. It is impossible for the server to accurately determine what might be "best" for any given user, since that would require complete knowledge of both the capabilities of the user agent and the intended use for the response (e.g., does the user want to view it on screen or print it on paper?).

2. Having the user agent describe its capabilities in every request can be both very inefficient (given that only a small percentage of responses have multiple representations) and a potential violation of the user's privacy.

3. It complicates the implementation of an origin server and the algorithms for generating responses to a request.

4. It might limit a public cache's ability to use the same response for multiple user's requests.

Server-driven negotiation allows the user agent to specify its preferences, but it cannot expect responses to always honor them. For example, the origin server might not implement server-driven negotiation, or it might decide that sending a response that doesn't conform to them is better than sending a 406 (Not Acceptable) response.

Many of the mechanisms for expressing preferences use quality values to declare relative preference. See Section 4.3.1 of [Part1] for

more information.

HTTP/1.1 includes the following header fields for enabling server-driven negotiation through description of user agent capabilities and user preferences: Accept (Section 9.1), Accept-Charset (Section 9.2), Accept-Encoding (Section 9.3), Accept-Language (Section 9.4), and User-Agent (Section 9.18). However, an origin server is not limited to these dimensions and MAY vary the response based on any aspect of the request, including aspects of the connection (e.g., IP address) or information within extension header fields not defined by this specification.

> Note: In practice, User-Agent based negotiation is fragile, because new clients might not be recognized.

The Vary header field (Section 7.5 of [Part6]) can be used to express the parameters the server uses to select a representation that is subject to server-driven negotiation.

8.2. Agent-driven Negotiation

With agent-driven negotiation, selection of the best representation for a response is performed by the user agent after receiving an initial response from the origin server. Selection is based on a list of the available representations of the response included within the header fields or body of the initial response, with each representation identified by its own URI. Selection from among the representations can be performed automatically (if the user agent is capable of doing so) or manually by the user selecting from a generated (possibly hypertext) menu.

Agent-driven negotiation is advantageous when the response would vary over commonly-used dimensions (such as type, language, or encoding), when the origin server is unable to determine a user agent's capabilities from examining the request, and generally when public caches are used to distribute server load and reduce network usage.

Agent-driven negotiation suffers from the disadvantage of needing a second request to obtain the best alternate representation. This second request is only efficient when caching is used. In addition, this specification does not define any mechanism for supporting automatic selection, though it also does not prevent any such mechanism from being developed as an extension and used within HTTP/1.1.

This specification defines the 300 (Multiple Choices) and 406 (Not Acceptable) status codes for enabling agent-driven negotiation when the server is unwilling or unable to provide a varying response using

server-driven negotiation.

9. Header Field Definitions

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

9.1. Accept

The "Accept" header field can be used by user agents to specify response media types that are acceptable. Accept header fields can be used to indicate that the request is specifically limited to a small set of desired types, as in the case of a request for an in-line image.

```
  Accept = #( media-range [ accept-params ] )

  media-range    = ( "*/*"
                   / ( type "/" "*" )
                   / ( type "/" subtype )
                   ) *( OWS ";" OWS parameter )
  accept-params  = OWS ";" OWS "q=" qvalue *( accept-ext )
  accept-ext     = OWS ";" OWS token [ "=" word ]
```

The asterisk "*" character is used to group media types into ranges, with "*/*" indicating all media types and "type/*" indicating all subtypes of that type. The media-range MAY include media type parameters that are applicable to that range.

Each media-range MAY be followed by one or more accept-params, beginning with the "q" parameter for indicating a relative quality factor. The first "q" parameter (if any) separates the media-range parameter(s) from the accept-params. Quality factors allow the user or user agent to indicate the relative degree of preference for that media-range, using the qvalue scale from 0 to 1 (Section 4.3.1 of [Part1]). The default value is q=1.

> Note: Use of the "q" parameter name to separate media type parameters from Accept extension parameters is due to historical practice. Although this prevents any media type parameter named "q" from being used with a media range, such an event is believed to be unlikely given the lack of any "q" parameters in the IANA media type registry and the rare usage of any media type parameters in Accept. Future media types are discouraged from registering any parameter named "q".

The example

```
  Accept: audio/*; q=0.2, audio/basic
```

SHOULD be interpreted as "I prefer audio/basic, but send me any audio type if it is the best available after an 80% mark-down in quality".

A request without any Accept header field implies that the user agent will accept any media type in response. If an Accept header field is present in a request and none of the available representations for the response have a media type that is listed as acceptable, the origin server MAY either honor the Accept header field by sending a 406 (Not Acceptable) response or disregard the Accept header field by treating the response as if it is not subject to content negotiation.

A more elaborate example is

```
  Accept: text/plain; q=0.5, text/html,
          text/x-dvi; q=0.8, text/x-c
```

Verbally, this would be interpreted as "text/html and text/x-c are the preferred media types, but if they do not exist, then send the text/x-dvi representation, and if that does not exist, send the text/plain representation".

Media ranges can be overridden by more specific media ranges or specific media types. If more than one media range applies to a given type, the most specific reference has precedence. For example,

```
  Accept: text/*, text/plain, text/plain;format=flowed, */*
```

have the following precedence:

1. text/plain;format=flowed
2. text/plain
3. text/*
4. */*

The media type quality factor associated with a given type is determined by finding the media range with the highest precedence which matches that type. For example,

```
  Accept: text/*;q=0.3, text/html;q=0.7, text/html;level=1,
          text/html;level=2;q=0.4, */*;q=0.5
```

would cause the following values to be associated:

Media Type	Quality Value
text/html;level=1	1
text/html	0.7
text/plain	0.3
image/jpeg	0.5
text/html;level=2	0.4
text/html;level=3	0.7

Note: A user agent might be provided with a default set of quality values for certain media ranges. However, unless the user agent is a closed system which cannot interact with other rendering agents, this default set ought to be configurable by the user.

9.2. Accept-Charset

The "Accept-Charset" header field can be used by user agents to indicate what character encodings are acceptable in a response payload. This field allows clients capable of understanding more comprehensive or special-purpose character encodings to signal that capability to a server which is capable of representing documents in those character encodings.

```
Accept-Charset = 1#( ( charset / "*" )
                     [ OWS ";" OWS "q=" qvalue ] )
```

Character encoding values (a.k.a., charsets) are described in Section 5.3. Each charset MAY be given an associated quality value which represents the user's preference for that charset. The default value is q=1. An example is

```
Accept-Charset: iso-8859-5, unicode-1-1;q=0.8
```

The special value "*", if present in the Accept-Charset field, matches every character encoding which is not mentioned elsewhere in the Accept-Charset field. If no "*" is present in an Accept-Charset field, then all character encodings not explicitly mentioned get a quality value of 0.

A request without any Accept-Charset header field implies that the user agent will accept any character encoding in response. If an Accept-Charset header field is present in a request and none of the available representations for the response have a character encoding that is listed as acceptable, the origin server MAY either honor the Accept-Charset header field by sending a 406 (Not Acceptable) response or disregard the Accept-Charset header field by treating the

response as if it is not subject to content negotiation.

9.3. Accept-Encoding

The "Accept-Encoding" header field can be used by user agents to indicate what response content-codings (Section 5.4) are acceptable in the response. An "identity" token is used as a synonym for "no encoding" in order to communicate when no encoding is preferred.

```
  Accept-Encoding  = #( codings [ OWS ";" OWS "q=" qvalue ] )
  codings          = content-coding / "identity" / "*"
```

Each codings value MAY be given an associated quality value which represents the preference for that encoding. The default value is q=1.

For example,

```
  Accept-Encoding: compress, gzip
  Accept-Encoding:
  Accept-Encoding: *
  Accept-Encoding: compress;q=0.5, gzip;q=1.0
  Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

A server tests whether a content-coding for a given representation is acceptable, according to an Accept-Encoding field, using these rules:

1. The special "*" symbol in an Accept-Encoding field matches any available content-coding not explicitly listed in the header field.

2. If the representation has no content-coding, then it is acceptable by default unless specifically excluded by the Accept-Encoding field stating either "identity;q=0" or "*;q=0" without a more specific entry for "identity".

3. If the representation’s content-coding is one of the content-codings listed in the Accept-Encoding field, then it is acceptable unless it is accompanied by a qvalue of 0. (As defined in Section 4.3.1 of [Part1], a qvalue of 0 means "not acceptable".)

4. If multiple content-codings are acceptable, then the acceptable content-coding with the highest non-zero qvalue is preferred.

An Accept-Encoding header field with a combined field-value that is empty implies that the user agent does not want any content-coding in response. If an Accept-Encoding header field is present in a request

and none of the available representations for the response have a content-coding that is listed as acceptable, the origin server SHOULD send a response without any content-coding.

A request without an Accept-Encoding header field implies that the user agent will accept any content-coding in response, but a representation without content-coding is preferred for compatibility with the widest variety of user agents.

> Note: Most HTTP/1.0 applications do not recognize or obey qvalues associated with content-codings. This means that qvalues will not work and are not permitted with x-gzip or x-compress.

9.4. Accept-Language

The "Accept-Language" header field can be used by user agents to indicate the set of natural languages that are preferred in the response. Language tags are defined in Section 5.6.

```
  Accept-Language =
                    1#( language-range [ OWS ";" OWS "q=" qvalue ] )
  language-range  =
            <language-range, defined in [RFC4647], Section 2.1>
```

Each language-range can be given an associated quality value which represents an estimate of the user's preference for the languages specified by that range. The quality value defaults to "q=1". For example,

```
  Accept-Language: da, en-gb;q=0.8, en;q=0.7
```

would mean: "I prefer Danish, but will accept British English and other types of English". (see also Section 2.3 of [RFC4647])

For matching, Section 3 of [RFC4647] defines several matching schemes. Implementations can offer the most appropriate matching scheme for their requirements.

> Note: The "Basic Filtering" scheme ([RFC4647], Section 3.3.1) is identical to the matching scheme that was previously defined in Section 14.4 of [RFC2616].

It might be contrary to the privacy expectations of the user to send an Accept-Language header field with the complete linguistic preferences of the user in every request. For a discussion of this issue, see Section 11.5.

As intelligibility is highly dependent on the individual user, it is

recommended that client applications make the choice of linguistic preference available to the user. If the choice is not made available, then the Accept-Language header field MUST NOT be given in the request.

> Note: When making the choice of linguistic preference available to the user, we remind implementers of the fact that users are not familiar with the details of language matching as described above, and ought to be provided appropriate guidance. As an example, users might assume that on selecting "en-gb", they will be served any kind of English document if British English is not available. A user agent might suggest in such a case to add "en" to get the best matching behavior.

9.5. 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.

9.6. Content-Encoding

The "Content-Encoding" header field indicates what content-codings have been applied to the representation beyond those inherent in the media type, and thus what decoding mechanisms have to be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a representation to be compressed without losing the identity of its underlying media type.

```
Content-Encoding = 1#content-coding
```

Content codings are defined in Section 5.4. An example of its use is

```
  Content-Encoding: gzip
```

The content-coding is a characteristic of the representation. Typically, the representation body is stored with this encoding and is only decoded before rendering or analogous usage. However, a transforming proxy MAY modify the content-coding if the new coding is known to be acceptable to the recipient, unless the "no-transform" cache-control directive is present in the message.

If the media type includes an inherent encoding, such as a data format that is always compressed, then that encoding would not be restated as a Content-Encoding even if it happens to be the same algorithm as one of the content-codings. Such a content-coding would only be listed if, for some bizarre reason, it is applied a second time to form the representation. Likewise, an origin server might choose to publish the same payload data as multiple representations that differ only in whether the coding is defined as part of Content-Type or Content-Encoding, since some user agents will behave differently in their handling of each response (e.g., open a "Save as ..." dialog instead of automatic decompression and rendering of content).

A representation that has a content-coding applied to it MUST include a Content-Encoding header field that lists the content-coding(s) applied.

If multiple encodings have been applied to a representation, the content codings MUST be listed in the order in which they were applied. Additional information about the encoding parameters MAY be provided by other header fields not defined by this specification.

If the content-coding of a representation in a request message is not acceptable to the origin server, the server SHOULD respond with a status code of 415 (Unsupported Media Type).

9.7. Content-Language

The "Content-Language" header field describes the natural language(s) of the intended audience for the representation. Note that this might not be equivalent to all the languages used within the representation.

```
  Content-Language = 1#language-tag
```

Language tags are defined in Section 5.6. The primary purpose of Content-Language is to allow a user to identify and differentiate representations according to the user’s own preferred language. Thus, if the body content is intended only for a Danish-literate

audience, the appropriate field is

```
Content-Language: da
```

If no Content-Language is specified, the default is that the content is intended for all language audiences. This might mean that the sender does not consider it to be specific to any natural language, or that the sender does not know for which language it is intended.

Multiple languages MAY be listed for content that is intended for multiple audiences. For example, a rendition of the "Treaty of Waitangi", presented simultaneously in the original Maori and English versions, would call for

```
Content-Language: mi, en
```

However, just because multiple languages are present within a representation does not mean that it is intended for multiple linguistic audiences. An example would be a beginner's language primer, such as "A First Lesson in Latin", which is clearly intended to be used by an English-literate audience. In this case, the Content-Language would properly only include "en".

Content-Language MAY be applied to any media type -- it is not limited to textual documents.

9.8. Content-Location

The "Content-Location" header field supplies a URI that can be used as a specific identifier for the representation in this message. In other words, if one were to perform a GET on this URI at the time of this message's generation, then a 200 (OK) response would contain the same representation that is enclosed as payload in this message.

```
Content-Location = absolute-URI / partial-URI
```

The Content-Location value is not a replacement for the effective Request URI (Section 5.5 of [Part1]). It is representation metadata. It has the same syntax and semantics as the header field of the same name defined for MIME body parts in Section 4 of [RFC2557]. However, its appearance in an HTTP message has some special implications for HTTP recipients.

If Content-Location is included in a response message and its value is the same as the effective request URI, then the response payload SHOULD be considered a current representation of that resource. For a GET or HEAD request, this is the same as the default semantics when no Content-Location is provided by the server. For a state-changing

request like PUT or POST, it implies that the server's response contains the new representation of that resource, thereby distinguishing it from representations that might only report about the action (e.g., "It worked!"). This allows authoring applications to update their local copies without the need for a subsequent GET request.

If Content-Location is included in a response message and its value differs from the effective request URI, then the origin server is informing recipients that this representation has its own, presumably more specific, identifier. For a GET or HEAD request, this is an indication that the effective request URI identifies a resource that is subject to content negotiation and the selected representation for this response can also be found at the identified URI. For other methods, such a Content-Location indicates that this representation contains a report on the action's status and the same report is available (for future access with GET) at the given URI. For example, a purchase transaction made via a POST request might include a receipt document as the payload of the 200 (OK) response; the Content-Location value provides an identifier for retrieving a copy of that same receipt in the future.

If Content-Location is included in a request message, then it MAY be interpreted by the origin server as an indication of where the user agent originally obtained the content of the enclosed representation (prior to any subsequent modification of the content by that user agent). In other words, the user agent is providing the same representation metadata that it received with the original representation. However, such interpretation MUST NOT be used to alter the semantics of the method requested by the client. For example, if a client makes a PUT request on a negotiated resource and the origin server accepts that PUT (without redirection), then the new set of values for that resource is expected to be consistent with the one representation supplied in that PUT; the Content-Location cannot be used as a form of reverse content selection that identifies only one of the negotiated representations to be updated. If the user agent had wanted the latter semantics, it would have applied the PUT directly to the Content-Location URI.

A Content-Location field received in a request message is transitory information that SHOULD NOT be saved with other representation metadata for use in later responses. The Content-Location's value might be saved for use in other contexts, such as within source links or other metadata.

A cache cannot assume that a representation with a Content-Location different from the URI used to retrieve it can be used to respond to later requests on that Content-Location URI.

If the Content-Location value is a partial URI, the partial URI is interpreted relative to the effective request URI.

9.9. Content-Type

The "Content-Type" header field indicates the media type of the representation. In the case of responses to the HEAD method, the media type is that which would have been sent had the request been a GET.

```
Content-Type = media-type
```

Media types are defined in Section 5.5. An example of the field is

```
Content-Type: text/html; charset=ISO-8859-4
```

Further discussion of Content-Type is provided in Section 7.3.

9.10. Date

The "Date" header field represents the date and time at which the message was originated, having the same semantics as the Origination Date Field (orig-date) defined in Section 3.6.1 of [RFC5322]. The field value is an HTTP-date, as defined in Section 5.1; it MUST be sent in rfc1123-date format.

```
Date = HTTP-date
```

An example is

```
Date: Tue, 15 Nov 1994 08:12:31 GMT
```

Origin servers MUST include a Date header field in all responses, except in these cases:

1. If the response status code is 100 (Continue) or 101 (Switching Protocols), the response MAY include a Date header field, at the server’s option.

2. If the response status code conveys a server error, e.g., 500 (Internal Server Error) or 503 (Service Unavailable), and it is inconvenient or impossible to generate a valid Date.

3. If the server does not have a clock that can provide a reasonable approximation of the current time, its responses MUST NOT include a Date header field.

A received message that does not have a Date header field MUST be

assigned one by the recipient if the message will be cached by that recipient.

Clients can use the Date header field as well; in order to keep request messages small, they are advised not to include it when it doesn't convey any useful information (as is usually the case for requests that do not contain a payload).

The HTTP-date sent in a Date header field SHOULD NOT represent a date and time subsequent to the generation of the message. It SHOULD represent the best available approximation of the date and time of message generation, unless the implementation has no means of generating a reasonably accurate date and time. In theory, the date ought to represent the moment just before the payload is generated. In practice, the date can be generated at any time during the message origination without affecting its semantic value.

9.11. Expect

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

```
  Expect       = 1#expectation

  expectation  = expect-name [ BWS "=" BWS expect-value ]
                             *( OWS ";" [ OWS expect-param ] )
  expect-param = expect-name [ BWS "=" BWS expect-value ]

  expect-name  = token
  expect-value = token / quoted-string
```

If all received Expect header field(s) are syntactically valid but contain an expectation that the recipient does not understand or cannot comply with, the recipient MUST respond with a 417 (Expectation Failed) status code. A recipient of a syntactically invalid Expectation header field MUST respond with a 4xx status code other than 417.

The only expectation defined by this specification is:

100-continue

> The "100-continue" expectation is defined Section 6.4.3 of [Part1]. It does not support any expect-params.

Comparison is case-insensitive for names (expect-name), and case-sensitive for values (expect-value).

The Expect mechanism is hop-by-hop: the above requirements apply to any server, including proxies. 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.

9.12. 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.

9.13. Location

The "Location" header field MAY be sent in responses to refer to a specific resource in accordance with the semantics of the status code.

```
  Location = URI-reference
```

For 201 (Created) responses, the Location is the URI of the new resource which was created by the request. For 3xx (Redirection) 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). If the original URI, as navigated to by the user agent, did contain a fragment identifier, and the final value does not, then the original URI's fragment identifier is added to the final value.

For example, the original URI "http://www.example.org/~tim", combined with a field value given as:

```
  Location: /pub/WWW/People.html#tim
```

would result in a final value of "http://www.example.org/pub/WWW/People.html#tim"

An original URI "http://www.example.org/index.html#larry", combined with a field value given as:

```
  Location: http://www.example.net/index.html
```

would result in a final value of "http://www.example.net/index.html#larry", preserving the original fragment identifier.

> Note: Some recipients attempt to recover from Location fields that are not valid URI references. This specification does not mandate or define such processing, but does allow it.

There are circumstances in which a fragment identifier in a Location URI would not be appropriate. For instance, when it appears in a 201 (Created) response, where the Location header field specifies the URI for the entire created resource.

> Note: The Content-Location header field (Section 9.8) 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.

9.14. Max-Forwards

The "Max-Forwards" header field provides a mechanism with the TRACE (Section 2.3.7) and OPTIONS (Section 2.3.1) 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 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.

9.15. Referer

The "Referer" [sic] header field allows the client to specify the URI of the resource from which the target 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 target 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.

9.16. 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 to 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.

9.17. 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 5.2) 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 6.2 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 implementers are encouraged to make this field a configurable option.

9.18. 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 5.2) 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	Idempotent	Reference
CONNECT	no	no	Section 2.3.8
DELETE	no	yes	Section 2.3.6
GET	yes	yes	Section 2.3.2
HEAD	yes	yes	Section 2.3.3
OPTIONS	yes	yes	Section 2.3.1
POST	no	no	Section 2.3.4
PUT	no	yes	Section 2.3.5
TRACE	yes	yes	Section 2.3.7

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 4.3.1
101	Switching Protocols	Section 4.3.2
200	OK	Section 4.4.1
201	Created	Section 4.4.2
202	Accepted	Section 4.4.3
203	Non-Authoritative Information	Section 4.4.4
204	No Content	Section 4.4.5
205	Reset Content	Section 4.4.6
300	Multiple Choices	Section 4.5.1
301	Moved Permanently	Section 4.5.2
302	Found	Section 4.5.3
303	See Other	Section 4.5.4
305	Use Proxy	Section 4.5.5
306	(Unused)	Section 4.5.6
307	Temporary Redirect	Section 4.5.7
400	Bad Request	Section 4.6.1
402	Payment Required	Section 4.6.2
403	Forbidden	Section 4.6.3
404	Not Found	Section 4.6.4
405	Method Not Allowed	Section 4.6.5
406	Not Acceptable	Section 4.6.6
408	Request Timeout	Section 4.6.7
409	Conflict	Section 4.6.8
410	Gone	Section 4.6.9
411	Length Required	Section 4.6.10
413	Request Representation Too Large	Section 4.6.11
414	URI Too Long	Section 4.6.12
415	Unsupported Media Type	Section 4.6.13
417	Expectation Failed	Section 4.6.14
426	Upgrade Required	Section 4.6.15
500	Internal Server Error	Section 4.7.1
501	Not Implemented	Section 4.7.2
502	Bad Gateway	Section 4.7.3
503	Service Unavailable	Section 4.7.4
504	Gateway Timeout	Section 4.7.5
505	HTTP Version Not Supported	Section 4.7.6

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
Accept	http	standard	Section 9.1
Accept-Charset	http	standard	Section 9.2
Accept-Encoding	http	standard	Section 9.3
Accept-Language	http	standard	Section 9.4
Allow	http	standard	Section 9.5
Content-Encoding	http	standard	Section 9.6
Content-Language	http	standard	Section 9.7
Content-Location	http	standard	Section 9.8
Content-Type	http	standard	Section 9.9
Date	http	standard	Section 9.10
Expect	http	standard	Section 9.11
From	http	standard	Section 9.12
Location	http	standard	Section 9.13
MIME-Version	http	standard	Appendix A.1
Max-Forwards	http	standard	Section 9.14
Referer	http	standard	Section 9.15
Retry-After	http	standard	Section 9.16
Server	http	standard	Section 9.17
User-Agent	http	standard	Section 9.18

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

10.4. Content Coding Registry

The registration procedure for HTTP Content Codings is now defined by Section 5.4.1 of this document.

The HTTP Content Codings Registry located at <http://www.iana.org/assignments/http-parameters> shall be updated with the registration below:

Name	Description	Reference
compress	UNIX "compress" program method	Section 4.2.1 of [Part1]
deflate	"deflate" compression mechanism ([RFC1951]) used inside the "zlib" data format ([RFC1950])	Section 4.2.2 of [Part1]
gzip	Same as GNU zip [RFC1952]	Section 4.2.3 of [Part1]

| identity | reserved (synonym for "no encoding" in Accept-Encoding header field) | Section 9.3 |

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. Implementers 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 9.18) or Server (Section 9.17) header fields can sometimes be used to determine that a specific client or server has 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 might contain enough entropy to be used, possibly in conjunction with other material, to uniquely identify the user.

Some request methods, like TRACE (Section 2.3.7), 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 Header Fields: Spoofing and Information Leakage

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.

Furthermore, appending the fragment identifier from one URI to another one obtained from a Location header field might leak confidential information to the target server -- although the fragment identifier is not transmitted in the final request, it might be visible to the user agent through other means, such as scripting.

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.

11.5. Privacy Issues Connected to Accept Header Fields

Accept header fields can reveal information about the user to all servers which are accessed. The Accept-Language header field in particular can reveal information the user would consider to be of a private nature, because the understanding of particular languages is often strongly correlated to the membership of a particular ethnic group. User agents which offer the option to configure the contents of an Accept-Language header field to be sent in every request are strongly encouraged to let the configuration process include a message which makes the user aware of the loss of privacy involved.

An approach that limits the loss of privacy would be for a user agent to omit the sending of Accept-Language header fields by default, and to ask the user whether or not to start sending Accept-Language header fields to a server if it detects, by looking for any Vary header fields generated by the server, that such sending could improve the quality of service.

Elaborate user-customized accept header fields sent in every request, in particular if these include quality values, can be used by servers as relatively reliable and long-lived user identifiers. Such user identifiers would allow content providers to do click-trail tracking, and would allow collaborating content providers to match cross-server click-trails or form submissions of individual users. Note that for many users not behind a proxy, the network address of the host running the user agent will also serve as a long-lived user identifier. In environments where proxies are used to enhance privacy, user agents ought to be conservative in offering accept header field configuration options to end users. As an extreme privacy measure, proxies could filter the accept header fields in relayed requests. General purpose user agents which provide a high degree of header field configurability SHOULD warn users about the loss of privacy which can be involved.

12. Acknowledgments

See Section 9 of [Part1].

13. References

13.1. Normative References

[Part1] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: Message Routing and Syntax"", draft-ietf-httpbis-p1-messaging-20 (work in progress), July 2012.

[Part4] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests", draft-ietf-httpbis-p4-conditional-20 (work in progress), July 2012.

[Part5] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests", draft-ietf-httpbis-p5-range-20 (work in progress), July 2012.

[Part6] Fielding, R., Ed., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-20 (work in progress), July 2012.

[Part7] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 7: Authentication", draft-ietf-httpbis-p7-auth-20 (work in progress), July 2012.

[RFC1950] Deutsch, L. and J-L. Gailly, "ZLIB Compressed Data Format Specification version 3.3", RFC 1950, May 1996.

[RFC1951] Deutsch, P., "DEFLATE Compressed Data Format Specification version 1.3", RFC 1951, May 1996.

[RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-

Pehrson, "GZIP file format specification version 4.3", RFC 1952, May 1996.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[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.

[RFC4647] Phillips, A., Ed. and M. Davis, Ed., "Matching of Language Tags", BCP 47, RFC 4647, September 2006.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, September 2009.

13.2. Informative References

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", STD 3, RFC 1123, October 1989.

[RFC1945] Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945,

May 1996.

[RFC2049] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples", RFC 2049, November 1996.

[RFC2068] Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2068, January 1997.

[RFC2076] Palme, J., "Common Internet Message Headers", RFC 2076, February 1997.

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC2295] Holtman, K. and A. Mutz, "Transparent Content Negotiation in HTTP", RFC 2295, March 1998.

[RFC2388] Masinter, L., "Returning Values from Forms: multipart/form-data", RFC 2388, August 1998.

[RFC2557] Palme, F., Hopmann, A., Shelness, N., and E. Stefferud, "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2557, March 1999.

[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.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

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

[RFC4288] Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures", BCP 13, RFC 4288, December 2005.

[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.

[RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, August 2010.

[RFC6151] Turner, S. and L. Chen, "Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms", RFC 6151, March 2011.

[RFC6266] Reschke, J., "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)", RFC 6266, June 2011.

[draft-reschke-http-status-308] Reschke, J., "The Hypertext Transfer Protocol (HTTP) Status Code 308 (Permanent Redirect)", draft-reschke-http-status-308-07 (work in progress), March 2012.

Appendix A. Differences between HTTP and MIME

HTTP/1.1 uses many of the constructs defined for Internet Mail ([RFC5322]) and the Multipurpose Internet Mail Extensions (MIME

[RFC2045]) to allow a message body to be transmitted in an open variety of representations and with extensible mechanisms. However, RFC 2045 discusses mail, and HTTP has a few features that are different from those described in MIME. These differences were carefully chosen to optimize performance over binary connections, to allow greater freedom in the use of new media types, to make date comparisons easier, and to acknowledge the practice of some early HTTP servers and clients.

This appendix describes specific areas where HTTP differs from MIME. Proxies and gateways to strict MIME environments SHOULD be aware of these differences and provide the appropriate conversions where necessary. Proxies and gateways from MIME environments to HTTP also need to be aware of the differences because some conversions might be required.

A.1. MIME-Version

HTTP is not a MIME-compliant protocol. However, HTTP/1.1 messages MAY include a single MIME-Version header field to indicate what version of the MIME protocol was used to construct the message. Use of the MIME-Version header field indicates that the message is in full conformance with the MIME protocol (as defined in [RFC2045]). Proxies/gateways are responsible for ensuring full conformance (where possible) when exporting HTTP messages to strict MIME environments.

```
  MIME-Version = 1*DIGIT "." 1*DIGIT
```

MIME version "1.0" is the default for use in HTTP/1.1. However, HTTP/1.1 message parsing and semantics are defined by this document and not the MIME specification.

A.2. Conversion to Canonical Form

MIME requires that an Internet mail body-part be converted to canonical form prior to being transferred, as described in Section 4 of [RFC2049]. Section 5.5.1 of this document describes the forms allowed for subtypes of the "text" media type when transmitted over HTTP. [RFC2046] requires that content with a type of "text" represent line breaks as CRLF and forbids the use of CR or LF outside of line break sequences. HTTP allows CRLF, bare CR, and bare LF to indicate a line break within text content when a message is transmitted over HTTP.

Where it is possible, a proxy or gateway from HTTP to a strict MIME environment SHOULD translate all line breaks within the text media types described in Section 5.5.1 of this document to the RFC 2049 canonical form of CRLF. Note, however, that this might be

complicated by the presence of a Content-Encoding and by the fact that HTTP allows the use of some character encodings which do not use octets 13 and 10 to represent CR and LF, respectively, as is the case for some multi-byte character encodings.

Conversion will break any cryptographic checksums applied to the original content unless the original content is already in canonical form. Therefore, the canonical form is recommended for any content that uses such checksums in HTTP.

A.3. Conversion of Date Formats

HTTP/1.1 uses a restricted set of date formats (Section 5.1) to simplify the process of date comparison. Proxies and gateways from other protocols SHOULD ensure that any Date header field present in a message conforms to one of the HTTP/1.1 formats and rewrite the date if necessary.

A.4. Introduction of Content-Encoding

MIME does not include any concept equivalent to HTTP/1.1's Content-Encoding header field. Since this acts as a modifier on the media type, proxies and gateways from HTTP to MIME-compliant protocols MUST either change the value of the Content-Type header field or decode the representation before forwarding the message. (Some experimental applications of Content-Type for Internet mail have used a media-type parameter of ";conversions=<content-coding>" to perform a function equivalent to Content-Encoding. However, this parameter is not part of the MIME standards).

A.5. No Content-Transfer-Encoding

HTTP does not use the Content-Transfer-Encoding field of MIME. Proxies and gateways from MIME-compliant protocols to HTTP MUST remove any Content-Transfer-Encoding prior to delivering the response message to an HTTP client.

Proxies and gateways from HTTP to MIME-compliant protocols are responsible for ensuring that the message is in the correct format and encoding for safe transport on that protocol, where "safe transport" is defined by the limitations of the protocol being used. Such a proxy or gateway SHOULD label the data with an appropriate Content-Transfer-Encoding if doing so will improve the likelihood of safe transport over the destination protocol.

A.6. MHTML and Line Length Limitations

HTTP implementations which share code with MHTML [RFC2557] implementations need to be aware of MIME line length limitations. Since HTTP does not have this limitation, HTTP does not fold long lines. MHTML messages being transported by HTTP follow all conventions of MHTML, including line length limitations and folding, canonicalization, etc., since HTTP transports all message-bodies as payload (see Section 5.5.2) and does not interpret the content or any MIME header lines that might be contained therein.

Appendix B. Additional Features

[RFC1945] and [RFC2068] document protocol elements used by some existing HTTP implementations, but not consistently and correctly across most HTTP/1.1 applications. Implementers are advised to be aware of these features, but cannot rely upon their presence in, or interoperability with, other HTTP/1.1 applications. Some of these describe proposed experimental features, and some describe features that experimental deployment found lacking that are now addressed in the base HTTP/1.1 specification.

A number of other header fields, such as Content-Disposition and Title, from SMTP and MIME are also often implemented (see [RFC6266] and [RFC2076]).

Appendix C. Changes from RFC 2616

Introduce Method Registry. (Section 2.2)

Clarify definition of POST. (Section 2.3.4)

Remove requirement to handle all Content-* header fields; ban use of Content-Range with PUT. (Section 2.3.5)

Take over definition of CONNECT method from [RFC2817]. (Section 2.3.8)

Take over the Status Code Registry, previously defined in Section 7.1 of [RFC2817]. (Section 4.2)

Broadened the definition of 203 (Non-Authoritative Information) to include cases of payload transformations as well. (Section 4.4.4)

Status codes 301, 302, and 307: removed the normative requirements on both response payloads and user interaction. (Section 4.5)

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. Furthermore, allow user agents to rewrite the method from POST to GET for status codes 301 and 302. (Sections 4.5.2, 4.5.3 and 4.5.7)

Deprecate 305 (Use Proxy) status code, because user agents did not implement it. It used to indicate that the target resource needs to 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 4.5.5)

Define status 426 (Upgrade Required) (this was incorporated from [RFC2817]). (Section 4.6.15)

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 9.5)

The ABNF for the Expect header field has been both fixed (allowing parameters for value-less expectations as well) and simplified (allowing trailing semicolons after "100-continue" when they were invalid before). (Section 9.11)

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 9.13)

Restrict Max-Forwards header field to OPTIONS and TRACE (previously, extension methods could have used it as well). (Section 9.14)

Allow Referer field value of "about:blank" as alternative to not specifying it. (Section 9.15)

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 6.2 of [Part1]. (Section 9.17)

Clarify contexts that charset is used in. (Section 5.3)

Registration of Content Codings now requires IETF Review (Section 5.4.1)

Remove the default character encoding of "ISO-8859-1" for text media types; the default now is whatever the media type definition says. (Section 5.5.1)

Change ABNF productions for header fields to only define the field value. (Section 9)

Remove definition of Content-MD5 header field because it was inconsistently implemented with respect to partial responses, and also because of known deficiencies in the hash algorithm itself (see [RFC6151] for details). (Section 9)

Remove ISO-8859-1 special-casing in Accept-Charset. (Section 9.2)

Remove base URI setting semantics for Content-Location due to poor implementation support, which was caused by too many broken servers emitting bogus Content-Location header fields, and also the potentially undesirable effect of potentially breaking relative links in content-negotiated resources. (Section 9.8)

Remove reference to non-existant identity transfer-coding value tokens. (Appendix A.5)

Remove discussion of Content-Disposition header field, it is now defined by [RFC6266]. (Appendix B)

Appendix D. Imported ABNF

The following core rules are included by reference, as defined in Appendix B.1 of [RFC5234]: 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), HTAB (horizontal tab), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), and VCHAR (any visible US-ASCII character).

The rules below are defined in [Part1]:

```
  BWS           = <BWS, defined in [Part1], Section 3.2.1>
  OWS           = <OWS, defined in [Part1], Section 3.2.1>
  RWS           = <RWS, defined in [Part1], Section 3.2.1>
  quoted-string = <quoted-string, defined in [Part1], Section 3.2.4>
  token         = <token, defined in [Part1], Section 3.2.4>
  word          = <word, defined in [Part1], Section 3.2.4>

  absolute-URI  = <absolute-URI, defined in [Part1], Section 2.8>
  comment       = <comment, defined in [Part1], Section 3.2.4>
  partial-URI   = <partial-URI, defined in [Part1], Section 2.8>
  qvalue        = <qvalue, defined in [Part1], Section 4.3.1>
```

```
     URI-reference = <URI-reference, defined in [Part1], Section 2.8>
```

Appendix E. Collected ABNF

```
   Accept = [ ( "," / ( media-range [ accept-params ] ) ) *( OWS "," [
    OWS ( media-range [ accept-params ] ) ] ) ]
   Accept-Charset = *( "," OWS ) ( ( charset / "*" ) [ OWS ";" OWS "q="
    qvalue ] ) *( OWS "," [ OWS ( ( charset / "*" ) [ OWS ";" OWS "q="
    qvalue ] ) ] )
   Accept-Encoding = [ ( "," / ( codings [ OWS ";" OWS "q=" qvalue ] ) )
    *( OWS "," [ OWS ( codings [ OWS ";" OWS "q=" qvalue ] ) ] ) ]
   Accept-Language = *( "," OWS ) ( language-range [ OWS ";" OWS "q="
    qvalue ] ) *( OWS "," [ OWS ( language-range [ OWS ";" OWS "q="
    qvalue ] ) ] )
   Allow = [ ( "," / method ) *( OWS "," [ OWS method ] ) ]

   BWS = <BWS, defined in [Part1], Section 3.2.1>

   Content-Encoding = *( "," OWS ) content-coding *( OWS "," [ OWS
    content-coding ] )
   Content-Language = *( "," OWS ) language-tag *( OWS "," [ OWS
    language-tag ] )
   Content-Location = absolute-URI / partial-URI
   Content-Type = media-type

   Date = HTTP-date

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

   From = mailbox

   GMT = %x47.4D.54 ; GMT

   HTTP-date = rfc1123-date / obs-date

   Location = URI-reference

   MIME-Version = 1*DIGIT "." 1*DIGIT
   Max-Forwards = 1*DIGIT

   OWS = <OWS, defined in [Part1], Section 3.2.1>

   RWS = <RWS, defined in [Part1], Section 3.2.1>
   Referer = absolute-URI / partial-URI
   Retry-After = HTTP-date / delta-seconds

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

```
   URI-reference = <URI-reference, defined in [Part1], Section 2.8>
   User-Agent = product *( RWS ( product / comment ) )

   absolute-URI = <absolute-URI, defined in [Part1], Section 2.8>
   accept-ext = OWS ";" OWS token [ "=" word ]
   accept-params = OWS ";" OWS "q=" qvalue *accept-ext
   asctime-date = day-name SP date3 SP time-of-day SP year
   attribute = token

   charset = token
   codings = content-coding / "identity" / "*"
   comment = <comment, defined in [Part1], Section 3.2.4>
   content-coding = token

   date1 = day SP month SP year
   date2 = day "-" month "-" 2DIGIT
   date3 = month SP ( 2DIGIT / ( SP DIGIT ) )
   day = 2DIGIT
   day-name = %x4D.6F.6E ; Mon
    / %x54.75.65 ; Tue
    / %x57.65.64 ; Wed
    / %x54.68.75 ; Thu
    / %x46.72.69 ; Fri
    / %x53.61.74 ; Sat
    / %x53.75.6E ; Sun
   day-name-l = %x4D.6F.6E.64.61.79 ; Monday
    / %x54.75.65.73.64.61.79 ; Tuesday
    / %x57.65.64.6E.65.73.64.61.79 ; Wednesday
    / %x54.68.75.72.73.64.61.79 ; Thursday
    / %x46.72.69.64.61.79 ; Friday
    / %x53.61.74.75.72.64.61.79 ; Saturday
    / %x53.75.6E.64.61.79 ; Sunday
   delta-seconds = 1*DIGIT

   expect-name = token
   expect-param = expect-name [ BWS "=" BWS expect-value ]
   expect-value = token / quoted-string
   expectation = expect-name [ BWS "=" BWS expect-value ] *( OWS ";" [
    OWS expect-param ] )

   hour = 2DIGIT

   language-range = <language-range, defined in [RFC4647], Section 2.1>
   language-tag = <Language-Tag, defined in [RFC5646], Section 2.1>

   mailbox = <mailbox, defined in [RFC5322], Section 3.4>
   media-range = ( "*/*" / ( type "/*" ) / ( type "/" subtype ) ) *( OWS
    ";" OWS parameter )
```

```
   media-type = type "/" subtype *( OWS ";" OWS parameter )
   method = token
   minute = 2DIGIT
   month = %x4A.61.6E ; Jan
    / %x46.65.62 ; Feb
    / %x4D.61.72 ; Mar
    / %x41.70.72 ; Apr
    / %x4D.61.79 ; May
    / %x4A.75.6E ; Jun
    / %x4A.75.6C ; Jul
    / %x41.75.67 ; Aug
    / %x53.65.70 ; Sep
    / %x4F.63.74 ; Oct
    / %x4E.6F.76 ; Nov
    / %x44.65.63 ; Dec

   obs-date = rfc850-date / asctime-date

   parameter = attribute "=" value
   partial-URI = <partial-URI, defined in [Part1], Section 2.8>
   product = token [ "/" product-version ]
   product-version = token

   quoted-string = <quoted-string, defined in [Part1], Section 3.2.4>
   qvalue = <qvalue, defined in [Part1], Section 4.3.1>

   rfc1123-date = day-name "," SP date1 SP time-of-day SP GMT
   rfc850-date = day-name-l "," SP date2 SP time-of-day SP GMT

   second = 2DIGIT
   subtype = token

   time-of-day = hour ":" minute ":" second
   token = <token, defined in [Part1], Section 3.2.4>
   type = token

   value = word

   word = <word, defined in [Part1], Section 3.2.4>

   year = 4DIGIT
```

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

F.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

F.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]

F.3. Since draft-ietf-httpbis-p3-payload-00

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/8>: "Media Type Registrations" (<http://purl.org/NET/http-errata#media-reg>)
- <http://tools.ietf.org/wg/httpbis/trac/ticket/14>: "Clarification regarding quoting of charset values" (<http://purl.org/NET/http-errata#charactersets>)
- <http://tools.ietf.org/wg/httpbis/trac/ticket/16>: "Remove 'identity' token references" (<http://purl.org/NET/http-errata#identity>)

- <http://tools.ietf.org/wg/httpbis/trac/ticket/25>: "Accept-Encoding BNF"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/35>: "Normative and Informative references"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/46>: "RFC1700 references"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/55>: "Updating to RFC4288"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/65>: "Informative references"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/66>: "ISO-8859-1 Reference"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/68>: "Encoding References Normative"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/86>: "Normative up-to-date references"

F.4. 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.

F.5. Since draft-ietf-httpbis-p3-payload-01

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

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

F.6. 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 field"
- <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 header fields 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).

F.7. Since draft-ietf-httpbis-p3-payload-02

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/67>: "Quoting Charsets"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/105>: "Classification for Allow header field"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/115>: "missing default for qvalue in description of Accept-Encoding"

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 header fields defined in this document.

F.8. 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).

F.9. Since draft-ietf-httpbis-p3-payload-03

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/67>: "Quoting Charsets"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/113>: "language tag matching (Accept-Language) vs RFC4647"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/121>: "RFC 1806 has been replaced by RFC2183"

Other changes:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/68>: "Encoding References Normative" -- rephrase the annotation and reference BCP97.

F.10. 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.

F.11. Since draft-ietf-httpbis-p3-payload-04

Closed issues:

- <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.

F.12. 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.

F.13. Since draft-ietf-httpbis-p3-payload-05

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/118>: "Join "Differences Between HTTP Entities and RFC 2045 Entities"?"

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

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

Other changes:

- Move definition of quality values into Part 1.

F.14. 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"

F.15. Since draft-ietf-httpbis-p3-payload-06

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/80>: "Content-Location isn't special"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/155>: "Content Sniffing"

F.16. 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 field 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?"

F.17. Since draft-ietf-httpbis-p3-payload-07

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/13>: "Updated reference for language tags"

- <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/154>: "Content-Location base-setting problems"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/155>: "Content Sniffing"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/188>: "pick IANA policy (RFC5226) for Transfer Coding / Content Coding"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/189>: "move definitions of gzip/deflate/compress to part 1"

Partly resolved issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/148>: "update IANA requirements wrt Transfer-Coding values" (add the IANA Considerations subsection)
- <http://tools.ietf.org/wg/httpbis/trac/ticket/149>: "update IANA requirements wrt Content-Coding values" (add the IANA Considerations subsection)

F.18. 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)

F.19. Since draft-ietf-httpbis-p3-payload-08

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/81>: "Content Negotiation for media types"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/181>: "Accept-Language: which RFC4647 filtering?"

F.20. 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 field payload handling"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/196>: "Term for the requested resource's URI"

F.21. Since draft-ietf-httpbis-p3-payload-09

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/122>: "MIME-Version not listed in P1, general header fields"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/143>: "IANA registry for content/transfer encodings"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/155>: "Content Sniffing"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/200>: "use of term "word" when talking about header field structure"

Partly resolved issues:

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

F.22. 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"

F.23. Since draft-ietf-httpbis-p3-payload-10

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/69>: "Clarify 'Requested Variant'"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/80>: "Content-Location isn't special"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/90>: "Delimiting messages with multipart/byteranges"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/109>: "Clarify entity / representation / variant terminology"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/136>: "confusing req. language for Content-Location"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/167>: "Content-Location on 304 responses"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/183>: "'requested resource' in content-encoding definition"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/220>: "consider removing the 'changes from 2068' sections"

Partly resolved issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/178>: "Content-MD5 and partial responses"

F.24. 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)

F.25. Since draft-ietf-httpbis-p3-payload-11

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/123>: "Factor out Content-Disposition"

F.26. 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 might 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 field 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 field 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 Field 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"

F.27. Since draft-ietf-httpbis-p3-payload-12

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/224>: "Header Field Classification"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/276>: "untangle ABNFs for header fields"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/277>: "potentially misleading MAY in media-type def"

F.28. 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"

F.29. Since draft-ietf-httpbis-p3-payload-13

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/20>: "Default charsets for text media types"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/178>: "Content-MD5 and partial responses"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/276>: "untangle ABNFs for header fields"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/281>: "confusing undefined parameter in media range example"

F.30. Since draft-ietf-httpbis-p2-semantics-14

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/255>: "Clarify status code for rate limiting"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/294>: "clarify 403 forbidden"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/296>: "Clarify 203 Non-Authoritative Information"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/298>: "update default reason phrase for 413"

F.31. Since draft-ietf-httpbis-p3-payload-14

None.

F.32. Since draft-ietf-httpbis-p2-semantics-15

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/285>: "Strength of requirements on Accept re: 406"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/303>: "400 response isn’t generic"

F.33. Since draft-ietf-httpbis-p3-payload-15

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/285>: "Strength of requirements on Accept re: 406"

F.34. Since draft-ietf-httpbis-p2-semantics-16

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/160>: "Redirects and non-GET methods"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/186>: "Document HTTP's error-handling philosophy"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/231>: "Considerations for new header fields"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/310>: "clarify 303 redirect on HEAD"

F.35. Since draft-ietf-httpbis-p3-payload-16

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/186>: "Document HTTP's error-handling philosophy"

F.36. Since draft-ietf-httpbis-p2-semantics-17

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/185>: "Location header field payload handling"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/255>: "Clarify status code for rate limiting" (change backed out because a new status code is being defined for this purpose)
- <http://tools.ietf.org/wg/httpbis/trac/ticket/312>: "should there be a permanent variant of 307"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/325>: "When are Location's semantics triggered?"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/327>: "'expect' grammar missing OWS"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/329>: "header field considerations: quoted-string vs use of double quotes"

F.37. Since draft-ietf-httpbis-p3-payload-17

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/323>: "intended maturity level vs normative references"

F.38. Since draft-ietf-httpbis-p2-semantics-18

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/227>: "Combining HEAD responses"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/238>: "Requirements for user intervention during redirects"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/250>: "message-body in CONNECT response"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/295>: "Applying original fragment to 'plain' redirected URI"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/302>: "Misplaced text on connection handling in p2"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/331>: "clarify that 201 doesn't require Location header fields"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/332>: "relax requirements on hypertext in 3/4/5xx error responses"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/333>: "example for 426 response should have a payload"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/336>: "drop indirection entries for status codes"

F.39. Since draft-ietf-httpbis-p3-payload-18

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/330>: "is ETag a representation header field?"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/338>: "Content-Location doesn't constrain the cardinality of representations"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/346>: "make IANA policy definitions consistent"

F.40. Since draft-ietf-httpbis-p2-semantics-19 and draft-ietf-httpbis-p3-payload-19

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/312>: "should there be a permanent variant of 307"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/347>: "clarify that 201 can imply *multiple* resources were created"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/351>: "merge P2 and P3"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/361>: "ABNF requirements for recipients"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/364>: "Capturing more information in the method registry"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/368>: "note introduction of new IANA registries as normative changes"

Index

Authors' Addresses

Roy T. Fielding (editor)
Adobe Systems Incorporated
345 Park Ave
San Jose, CA 95110
USA

EMail: fielding@gbiv.com
URI: http://roy.gbiv.com/

Yves Lafon (editor)
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)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

EMail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/