Network Working Group	R. Fielding, Ed.
Internet-Draft	Day Software
Obsoletes: 2616 (if approved)	J. Gettys
Updates: 2817 (if approved)	One Laptop per Child
Intended status: Standards Track	J. Mogul
Expires: December 19, 2008	HP
	H. Frystyk
	Microsoft
	L. Masinter
	Adobe Systems
	P. Leach
	Microsoft
	T. Berners-Lee
	W3C/MIT
	Y. Lafon, Ed.
	W3C
	J. Reschke, Ed.
	greenbytes
	June 17, 2008

TOC

HTTP/1.1, part 2: Message Semantics
draft-ietf-httpbis-p2-semantics-03

Status of this Memo

This Internet-Draft will expire on December 19, 2008.

Abstract

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

Editorial Note (To be removed by RFC Editor)

Discussion of this draft should take place on the HTTPBIS working group mailing list (ietf-http-wg@w3.org). The current issues list is at http://www.tools.ietf.org/wg/httpbis/trac/report/11 and related documents (including fancy diffs) can be found at http://www.tools.ietf.org/wg/httpbis/.

The changes in this draft are summarized in Appendix B.4 (Since draft-ietf-httpbis-p2-semantics-02).

Table of Contents

1. Introduction

TOC

This document defines HTTP/1.1 request and response semantics. Each HTTP message, as defined in [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” June 2008.), is in the form of either a request or a response. An HTTP server listens on a connection for HTTP requests and responds to each request, in the order received on that connection, with one or more HTTP response messages. This document defines the commonly agreed upon semantics of the HTTP uniform interface, the intentions defined by each request method, and the various response messages that might be expected as a result of applying that method for the requested resource.

This document is currently disorganized in order to minimize the changes between drafts and enable reviewers to see the smaller errata changes. The next 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, general header fields, methods, request modifiers, response status, and resource metadata. The current mess reflects how widely dispersed these topics and associated requirements had become in [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.).

1.1. Requirements

TOC

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] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).

An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant."

---

2. Notational Conventions and Generic Grammar

TOC

This specification uses the ABNF syntax defined in Section 2.1 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” June 2008.) and the core rules defined in Section 2.2 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” June 2008.): [abnf.dep] (ABNF syntax and basic rules will be adopted from RFC 5234, see <http://tools.ietf.org/wg/httpbis/trac/ticket/36>.)

```
    DIGIT         = <DIGIT, defined in [Part1], Section 2.2>


    comment       = <comment, defined in [Part1], Section 2.2>
    quoted-string = <quoted-string, defined in [Part1], Section 2.2>
    token         = <token, defined in [Part1], Section 2.2>
```

The ABNF rules below are defined in other parts:

```
    absoluteURI   = <absoluteURI, defined in [Part1], Section 3.2.1>
    fragment      = <fragment, defined in [Part1], Section 3.2.1>
    Host          = <Host, defined in [Part1], Section 8.4>
    HTTP-date     = <HTTP-date, defined in [Part1], Section 3.3.1>
    product       = <product, defined in [Part1], Section 3.5>
    relativeURI   = <relativeURI, defined in [Part1], Section 3.2.1>
    TE            = <TE, defined in [Part1], Section 8.8>
```

```
Accept        = <Accept, defined in [Part3], Section 6.1>
Accept-Charset =
           <Accept-Charset, defined in [Part3], Section 6.2>
Accept-Encoding =
           <Accept-Encoding, defined in [Part3], Section 6.3>
Accept-Language =
           <Accept-Language, defined in [Part3], Section 6.4>


ETag          = <ETag, defined in [Part4], Section 7.1>
If-Match      = <If-Match, defined in [Part4], Section 7.2>
If-Modified-Since =
           <If-Modified-Since, defined in [Part4], Section 7.3>
If-None-Match = <If-None-Match, defined in [Part4], Section 7.4>
If-Unmodified-Since =
           <If-Unmodified-Since, defined in [Part4], Section 7.5>


Accept-Ranges = <Accept-Ranges, defined in [Part5], Section 6.1>
If-Range      = <If-Range, defined in [Part5], Section 6.3>
Range         = <Range, defined in [Part5], Section 6.4>


Age           = <Age, defined in [Part6], Section 16.1>
Vary          = <Vary, defined in [Part6], Section 16.5>


Authorization = <Authorization, defined in [Part7], Section 4.1>
Proxy-Authenticate =
           <Proxy-Authenticate, defined in [Part7], Section 4.2>
Proxy-Authorization =
           <Proxy-Authorization, defined in [Part7], Section 4.3>
WWW-Authenticate =
           <WWW-Authenticate, defined in [Part7], Section 4.4>
```

---

3. Method

TOC

The Method token indicates the method to be performed on the resource identified by the Request-URI. The method is case-sensitive.

```
Method         = %x4F.50.54.49.4F.4E.53   ; "OPTIONS", Section 8.2
               | %x47.45.54               ; "GET", Section 8.3
               | %x48.45.41.44            ; "HEAD", Section 8.4
               | %x50.4F.53.54            ; "POST", Section 8.5
               | %x50.55.54               ; "PUT", Section 8.6
               | %x44.45.4C.45.54.45      ; "DELETE", Section 8.7
               | %x54.52.41.43.45         ; "TRACE", Section 8.8
               | %x43.4F.4E.4E.45.43.54   ; "CONNECT", Section 8.9
               | extension-method
extension-method = token
```

The list of methods allowed by a resource can be specified in an Allow header field (Section 10.1 (Allow)). The return 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 return the status code 405 (Method Not Allowed) if the method is known by the origin server but not allowed for the requested 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 8 (Method Definitions).

---

4. Request Header Fields

TOC

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.

```
    request-header = Accept                   ; [Part3], Section 6.1
                   | Accept-Charset           ; [Part3], Section 6.2
                   | Accept-Encoding          ; [Part3], Section 6.3
                   | Accept-Language          ; [Part3], Section 6.4
                   | Authorization            ; [Part7], Section 4.1
                   | Expect                   ; Section 10.2
                   | From                     ; Section 10.3
                   | Host                     ; [Part1], Section 8.4
                   | If-Match                 ; [Part4], Section 7.2
                   | If-Modified-Since        ; [Part4], Section 7.3
                   | If-None-Match            ; [Part4], Section 7.4
                   | If-Range                 ; [Part5], Section 6.3
                   | If-Unmodified-Since      ; [Part4], Section 7.5
                   | Max-Forwards             ; Section 10.5
                   | Proxy-Authorization      ; [Part7], Section 4.3
                   | Range                    ; [Part5], Section 6.4
                   | Referer                  ; Section 10.6
                   | TE                       ; [Part1], Section 8.8
                   | User-Agent               ; Section 10.9
```

Request-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of request-header fields if all parties in the communication recognize them to be request-header fields. Unrecognized header fields are treated as entity-header fields.

TOC

5. Status Code and Reason Phrase

The Status-Code element is a 3-digit integer result code of the attempt to understand and satisfy the request. The status codes listed below are defined in Section 9 (Status Code Definitions). The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata and the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

The individual values of the numeric status codes defined for HTTP/1.1, and an example set of corresponding Reason-Phrase's, are presented below. The reason phrases listed here are only recommendations -- they MAY be replaced by local equivalents without affecting the protocol.

```
    Status-Code    =
           "100"  ; Section 9.1.1: Continue
         | "101"  ; Section 9.1.2: Switching Protocols
         | "200"  ; Section 9.2.1: OK
         | "201"  ; Section 9.2.2: Created
         | "202"  ; Section 9.2.3: Accepted
         | "203"  ; Section 9.2.4: Non-Authoritative Information
         | "204"  ; Section 9.2.5: No Content
         | "205"  ; Section 9.2.6: Reset Content
         | "206"  ; Section 9.2.7: Partial Content
         | "300"  ; Section 9.3.1: Multiple Choices
         | "301"  ; Section 9.3.2: Moved Permanently
         | "302"  ; Section 9.3.3: Found
         | "303"  ; Section 9.3.4: See Other
         | "304"  ; Section 9.3.5: Not Modified
         | "305"  ; Section 9.3.6: Use Proxy
         | "307"  ; Section 9.3.8: Temporary Redirect
         | "400"  ; Section 9.4.1: Bad Request
         | "401"  ; Section 9.4.2: Unauthorized
         | "402"  ; Section 9.4.3: Payment Required
         | "403"  ; Section 9.4.4: Forbidden
         | "404"  ; Section 9.4.5: Not Found
         | "405"  ; Section 9.4.6: Method Not Allowed
         | "406"  ; Section 9.4.7: Not Acceptable
         | "407"  ; Section 9.4.8: Proxy Authentication Required
         | "408"  ; Section 9.4.9: Request Time-out
         | "409"  ; Section 9.4.10: Conflict
         | "410"  ; Section 9.4.11: Gone
         | "411"  ; Section 9.4.12: Length Required
         | "412"  ; Section 9.4.13: Precondition Failed
         | "413"  ; Section 9.4.14: Request Entity Too Large
         | "414"  ; Section 9.4.15: Request-URI Too Large
         | "415"  ; Section 9.4.16: Unsupported Media Type
         | "416"  ; Section 9.4.17: Requested range not satisfiable
         | "417"  ; Section 9.4.18: Expectation Failed
         | "500"  ; Section 9.5.1: Internal Server Error
         | "501"  ; Section 9.5.2: Not Implemented
         | "502"  ; Section 9.5.3: Bad Gateway
         | "503"  ; Section 9.5.4: Service Unavailable
         | "504"  ; Section 9.5.5: Gateway Time-out
         | "505"  ; Section 9.5.6: HTTP Version not supported
         | extension-code

    extension-code = 3DIGIT
    Reason-Phrase  = *<TEXT, excluding CR, LF>
```

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 entity returned with the response, since that entity is likely to include human-readable information which will explain the unusual status.

5.1. Status Code Registry

TOC

The HTTP Status Code Registry defines the name space for the Status-Code token in the Status line of an HTTP response.

Values to be added to this name space are subject to IETF review ([RFC5226] (Narten, T. and H. Alvestrand, “Guidelines for Writing an IANA Considerations Section in RFCs,” May 2008.), Section 4.1). Any document registering new status codes should be traceable through statuses of either 'Obsoletes' or 'Updates' to this document.

The registry itself is maintained at http://www.iana.org/assignments/http-status-codes.

6. Response Header Fields

TOC

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 resource identified by the Request-URI.

```
response-header = Accept-Ranges           ; [Part5], Section 6.1
                | Age                     ; [Part6], Section 16.1
                | Allow                   ; Section 10.1
                | ETag                    ; [Part4], Section 7.1
                | Location                ; Section 10.4
                | Proxy-Authenticate      ; [Part7], Section 4.2
                | Retry-After             ; Section 10.7
                | Server                  ; Section 10.8
                | Vary                    ; [Part6], Section 16.5
                | WWW-Authenticate        ; [Part7], Section 4.4
```

Response-header field names can be extended reliably only in combination with a change in the protocol version. However, new or experimental header fields MAY be given the semantics of response-header fields if all parties in the communication recognize them to be response-header fields. Unrecognized header fields are treated as entity-header fields.

TOC

7. Entity

Request and Response messages MAY transfer an entity if not otherwise restricted by the request method or response status code. An entity consists of entity-header fields and an entity-body, although some responses will only include the entity-headers. HTTP entity-body and entity-header fields are defined in [Part3] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 3: Message Payload and Content Negotiation," June 2008.).

An entity-body is only present in a message when a message-body is present, as described in Section 4.3 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," June 2008.). The entity-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.

---

8. Method Definitions

TOC

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

8.1. Safe and Idempotent Methods

TOC

8.1.1. Safe Methods

TOC

Implementors should be aware that the software represents the user in their interactions over the Internet, and should be careful to allow the user to be aware of any actions they might take which may have an unexpected significance to themselves or others.

In particular, the convention has been established that the GET and HEAD methods SHOULD NOT have the significance of taking an action other than retrieval. These 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.

8.1.2. Idempotent Methods

TOC

Methods can also have the property of "idempotence" in that (aside from error or expiration issues) the side-effects of N > 0 identical requests is the same as for a single request. The methods GET, HEAD, PUT and DELETE share this property. Also, the methods OPTIONS and TRACE SHOULD NOT have side effects, and so are inherently idempotent.

However, it is possible that a sequence of several requests is non-idempotent, even if all of the methods executed in that sequence are idempotent. (A sequence is idempotent if a single execution of the entire sequence always yields a result that is not changed by a reexecution of all, or part, of that sequence.) For example, a sequence is non-idempotent if its result depends on a value that is later modified in the same sequence.

A sequence that never has side effects is idempotent, by definition (provided that no concurrent operations are being executed on the same set of resources).

8.2. OPTIONS

TOC

The OPTIONS method represents a request for information about the communication options available on the request/response chain identified by the Request-URI. This method allows the 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 this method are not cacheable.

If the OPTIONS request includes an entity-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. A server that does not support such an extension MAY discard the request body.

If the Request-URI is an asterisk ("*"), the OPTIONS request is intended to apply to the server in general rather than to a specific resource. Since a server's communication options typically depend on the resource, the "*" request is only useful as a "ping" or "no-op" type of method; it does nothing beyond allowing the client to test

the capabilities of the server. For example, this can be used to test a proxy for HTTP/1.1 compliance (or lack thereof).

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

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

The Max-Forwards request-header field MAY be used to target a specific proxy in the request chain. When a proxy receives an OPTIONS request on an absoluteURI for which request forwarding is permitted, the proxy MUST check for a Max-Forwards field. If the Max-Forwards field-value is zero ("0"), the proxy MUST NOT forward the message; instead, the proxy SHOULD respond with its own communication options. If the Max-Forwards field-value is an integer greater than zero, the proxy MUST decrement the field-value when it forwards the request. If no Max-Forwards field is present in the request, then the forwarded request MUST NOT include a Max-Forwards field.

8.3. GET

TOC

The GET method means retrieve whatever information (in the form of an entity) is identified by the Request-URI. If the Request-URI refers to a data-producing process, it is the produced data which shall be returned as the entity in the response and not the source text of the process, unless that text happens to be the output of the process.

The semantics of the GET method change to a "conditional GET" if the request message includes an If-Modified-Since, If-Unmodified-Since, If-Match, If-None-Match, or If-Range header field. A conditional GET method requests that the entity be transferred only under the circumstances described by the conditional header field(s). The conditional GET method is intended to reduce unnecessary network usage by allowing cached entities to be refreshed without requiring multiple requests or transferring data already held by the client.

The semantics of the GET method change to a "partial GET" if the request message includes a Range header field. A partial GET requests that only part of the entity be transferred, as described in Section 6.4 of [Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y.,

Ed., and J. Reschke, Ed., “HTTP/1.1, part 5: Range Requests and Partial Responses,” June 2008.). The partial GET method is intended to reduce unnecessary network usage by allowing partially-retrieved entities to be completed without transferring data already held by the client.

The response to a GET request is cacheable if and only if it meets the requirements for HTTP caching described in [Part6] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 6: Caching,” June 2008.).

See Section 12.2 (Encoding Sensitive Information in URIs) for security considerations when used for forms.

8.4. HEAD

TOC

The HEAD method is identical to GET except that the server MUST NOT return a message-body in the response. The metainformation contained in the HTTP headers 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 metainformation about the entity implied by the request without transferring the entity-body itself. This method is often used for testing hypertext links for validity, accessibility, and recent modification.

The response to a HEAD request MAY be cacheable in the sense that the information contained in the response MAY be used to update a previously cached entity from that resource. If the new field values indicate that the cached entity differs from the current entity (as would be indicated by a change in Content-Length, Content-MD5, ETag or Last-Modified), then the cache MUST treat the cache entry as stale.

8.5. POST

TOC

The POST method is used to request that the origin server accept the entity enclosed in the request as data to be processed by the resource identified by the Request-URI in the Request-Line. 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 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, depending on whether or not the response includes an entity that describes the result.

If a resource has been created on the origin server, the response SHOULD be 201 (Created) and contain an entity which describes the status of the request and refers to the new resource, and a Location header (see Section 10.4 (Location)).

Responses to this method are not cacheable, unless the response includes appropriate Cache-Control or Expires header fields. However, the 303 (See Other) response can be used to direct the user agent to retrieve a cacheable resource.

8.6. PUT

TOC

The PUT method requests that the enclosed entity be stored at the supplied Request-URI. If the Request-URI refers to an already existing resource, the enclosed entity SHOULD be considered as a modified version of the one residing on the origin server. If the Request-URI does not point to an existing resource, and that URI is capable of being defined as a new resource by the requesting user agent, the origin server can create the resource with that URI. If a new resource is created at the Request-URI, the origin server MUST inform the user agent via the 201 (Created) response. If an existing resource is modified, either the 200 (OK) or 204 (No Content) response codes SHOULD be sent to indicate successful completion of the request. If the resource could not be created or modified with the Request-URI, an appropriate error response SHOULD be given that reflects the nature of the problem. The recipient of the entity MUST NOT ignore any Content-* (e.g. Content-Range) headers that it does not understand or implement and MUST return a 501 (Not Implemented) response in such cases.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

The fundamental difference between the POST and PUT requests is reflected in the different meaning of the Request-URI. The URI in a POST request identifies the resource that will handle the enclosed entity. That resource might be a data-accepting process, a gateway to some other protocol, or a separate entity that accepts annotations. In contrast, the URI in a PUT request identifies the entity enclosed with the request -- the user agent knows what URI is intended and the server MUST NOT attempt to apply the request to

some other resource. If the server desires that the request be applied to a different URI, it 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 single resource MAY be identified by many different URIs. For example, an article might have a URI for identifying "the current version" which is separate from the URI identifying each particular version. In this case, a PUT request on a general URI might result in several other URIs being defined by the origin server.

HTTP/1.1 does not define how a PUT method affects the state of an origin server.

Unless otherwise specified for a particular entity-header, the entity-headers in the PUT request SHOULD be applied to the resource created or modified by the PUT.

8.7. DELETE

TOC

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

A successful response SHOULD be 200 (OK) if the response includes an entity 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 an entity.

If the request passes through a cache and the Request-URI identifies one or more currently cached entities, those entries SHOULD be treated as stale. Responses to this method are not cacheable.

8.8. TRACE

TOC

The TRACE method is used to invoke 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 entity-body of a 200 (OK) response. The final recipient is either the origin server or the first proxy or gateway to receive a Max-Forwards value of zero (0) in the request (see Section 10.5 (Max-Forwards)). A TRACE request MUST NOT include an entity.

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 8.9 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," June 2008.)) 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 contain the entire request message in the entity-body, with a Content-Type of "message/http" (see Appendix A.1 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," June 2008.)). Responses to this method MUST NOT be cached.

8.9. CONNECT

TOC

This specification reserves the method name CONNECT for use with a proxy that can dynamically switch to being a tunnel (e.g. SSL tunneling [Luo1998] (Luotonen, A., "Tunneling TCP based protocols through Web proxy servers," August 1998.)).

---

9. Status Code Definitions

TOC

Each Status-Code is described below, including a description of which method(s) it can follow and any metainformation required in the response.

9.1. Informational 1xx

TOC

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

A client MUST be prepared to accept one or more 1xx status responses prior to a regular response, even if the client does not expect a

100 (Continue) status message. Unexpected 1xx status responses MAY be ignored by a user agent.

Proxies MUST forward 1xx responses, unless the connection between the proxy and its client has been closed, or unless the proxy itself requested the generation of the 1xx response. (For example, if a proxy adds a "Expect: 100-continue" field when it forwards a request, then it need not forward the corresponding 100 (Continue) response(s).)

9.1.1. 100 Continue

TOC

The client SHOULD continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response. The server MUST send a final response after the request has been completed. See Section 7.2.3 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” June 2008.) for detailed discussion of the use and handling of this status code.

9.1.2. 101 Switching Protocols

TOC

The server understands and is willing to comply with the client's request, via the Upgrade message header field (Section 6.4 of [Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 5: Range Requests and Partial Responses,” June 2008.)), 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.

TOC

9.2. Successful 2xx

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

9.2.1. 200 OK

TOC

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

> **GET** an entity corresponding to the requested resource is sent in the response;
>
> **HEAD** the entity-header fields corresponding to the requested resource are sent in the response without any message-body;
>
> **POST** an entity describing or containing the result of the action;
>
> **TRACE** an entity containing the request message as received by the end server.

9.2.2. 201 Created

TOC

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

A 201 response MAY contain an ETag response header field indicating the current value of the entity tag for the requested variant just created, see Section 7.1 of [Part4] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 4: Conditional Requests,” June 2008.).

9.2.3. 202 Accepted

TOC

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 entity 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.

9.2.4. 203 Non-Authoritative Information

TOC

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

9.2.5. 204 No Content

TOC

The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metainformation. The response MAY include new or updated metainformation in the form of entity-headers, which if present SHOULD be associated with the requested variant.

If the client is a user agent, it SHOULD NOT change its document view from that which caused the request to be sent. This response is primarily intended to allow input for actions to take place without causing a change to the user agent's active document view, although any new or updated metainformation SHOULD be applied to the document currently in the user agent's active view.

The 204 response MUST NOT include a message-body, and thus is always terminated by the first empty line after the header fields.

9.2.6. 205 Reset Content

TOC

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 response MUST NOT include an entity.

9.2.7. 206 Partial Content

TOC

The server has fulfilled the partial GET request for the resource and the enclosed entity is a partial representation as defined in [Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 5: Range Requests and Partial Responses,” June 2008.).

9.3. Redirection 3xx

TOC

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

> Note: previous versions of this specification recommended a maximum of five redirections. Content developers should be aware that there might be clients that implement such a fixed limitation.

9.3.1. 300 Multiple Choices

TOC

The requested resource corresponds to any one of a set of representations, each with its own specific location, and agent-driven negotiation information (Section 5 of [Part3] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1,

part 3: Message Payload and Content Negotiation," June 2008.)) is being provided so that the user (or user agent) can select a preferred representation and redirect its request to that location.

Unless it was a HEAD request, the response SHOULD include an entity containing a list of resource characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

If the server has a preferred choice of representation, it SHOULD include the specific URI for that representation in the Location field; user agents MAY use the Location field value for automatic redirection. This response is cacheable unless indicated otherwise.

9.3.2. 301 Moved Permanently

TOC

The requested 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 Request-URI to one or more of the new references returned by the server, where possible. This response is cacheable unless indicated otherwise.

The new permanent URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 301 status code is received in response to a request method that is known to be "safe", as defined in Section 8.1.1 (Safe Methods), then the request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

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

9.3.3. 302 Found

TOC

The requested resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client

SHOULD continue to use the Request-URI for future requests. This response is only cacheable if indicated by a Cache-Control or Expires header field.

The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s).

If the 302 status code is received in response to a request method that is known to be "safe", as defined in Section 8.1.1 (Safe Methods), then the request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

> Note: [RFC1945] (Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0," May 1996.) and [RFC2068] (Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," January 1997.) specify that the client is not allowed to change the method on the redirected request. However, most existing user agent implementations treat 302 as if it were a 303 response, performing a GET on the Location field-value regardless of the original request method. The status codes 303 and 307 have been added for servers that wish to make unambiguously clear which kind of reaction is expected of the client.

9.3.4. 303 See Other

TOC

The server directs the user agent to a different resource, indicated by a URI in the Location header field, that provides an indirect response to the original request. The user agent MAY perform a GET request on the URI in the Location field in order to obtain a representation corresponding to the response, be redirected again, or end with an error status. The Location URI is not a substitute reference for the originally requested resource.

The 303 status 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 requested resource such that the follow-on representation may be useful without implying that it adequately represents the previously requested 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 resource owner(s).

A 303 response SHOULD NOT be cached unless it is indicated as cacheable by Cache-Control or Expires header fields. Except for responses to a HEAD request, the entity of a 303 response SHOULD contain a short hypertext note with a hyperlink to the Location URI.

9.3.5. 304 Not Modified

TOC

The response to the request has not been modified since the conditions indicated by the client's conditional GET request, as defined in [Part4] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests," June 2008.).

9.3.6. 305 Use Proxy

TOC

The 305 status was defined in a previous version of this specification (see Appendix A.2 (Changes from RFC 2616)), and is now deprecated.

9.3.7. 306 (Unused)

TOC

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

9.3.8. 307 Temporary Redirect

TOC

The requested resource resides temporarily under a different URI. Since the redirection MAY be altered on occasion, the client SHOULD continue to use the Request-URI for future requests. This response is only cacheable if indicated by a Cache-Control or Expires header field.

The temporary URI SHOULD be given by the Location field in the response. Unless the request method was HEAD, the entity of the response SHOULD contain a short hypertext note with a hyperlink to the new URI(s) , since many pre-HTTP/1.1 user agents do not understand the 307 status. Therefore, the note SHOULD contain the information necessary for a user to repeat the original request on the new URI.

If the 307 status code is received in response to a request method that is known to be "safe", as defined in Section 8.1.1 (Safe Methods), then the request MAY be automatically redirected by the user agent without confirmation. Otherwise, the user agent MUST NOT automatically redirect the request unless it can be confirmed by the user, since this might change the conditions under which the request was issued.

9.4. Client Error 4xx

TOC

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 an entity 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 entity to the user.

If the client is sending data, a server implementation using TCP SHOULD be careful to ensure that the client acknowledges receipt of the packet(s) containing the response, before the server closes the input connection. If the client continues sending data to the server after the close, the server's TCP stack will send a reset packet to the client, which may erase the client's unacknowledged input buffers before they can be read and interpreted by the HTTP application.

9.4.1. 400 Bad Request

TOC

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

9.4.2. 401 Unauthorized

TOC

The request requires user authentication (see [Part7] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 7: Authentication,” June 2008.)).

9.4.3. 402 Payment Required

TOC

This code is reserved for future use.

9.4.4. 403 Forbidden

TOC

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

9.4.5. 404 Not Found

TOC

The server has not found anything matching the 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.

9.4.6. 405 Method Not Allowed

TOC

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

9.4.7. 406 Not Acceptable

TOC

The resource identified by the request is only capable of generating response entities which have content characteristics not acceptable according to the accept headers sent in the request.

Unless it was a HEAD request, the response SHOULD include an entity containing a list of available entity characteristics and location(s) from which the user or user agent can choose the one most appropriate. The entity format is specified by the media type given in the Content-Type header field. Depending upon the format and the capabilities of the user agent, selection of the most appropriate choice MAY be performed automatically. However, this specification does not define any standard for such automatic selection.

> Note: HTTP/1.1 servers are allowed to return responses which are not acceptable according to the accept headers sent in the request. In some cases, this may even be preferable to sending a 406 response. User agents are encouraged to inspect the headers 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.

9.4.8. 407 Proxy Authentication Required

TOC

This code is similar to 401 (Unauthorized), but indicates that the client must first authenticate itself with the proxy (see [Part7] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 7: Authentication,” June 2008.)).

9.4.9. 408 Request Timeout

TOC

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.

TOC

9.4.10. 409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough information for the user to recognize the source of the conflict. Ideally, the response entity 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 entity 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 entity would likely contain a list of the differences between the two versions in a format defined by the response Content-Type.

9.4.11. 410 Gone

TOC

The requested 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 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. This response is cacheable unless indicated otherwise.

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.

9.4.12. 411 Length Required

TOC

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.

9.4.13. 412 Precondition Failed

TOC

The precondition given in one or more of the request-header fields evaluated to false when it was tested on the server, as defined in [Part4] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 4: Conditional Requests,” June 2008.).

9.4.14. 413 Request Entity Too Large

TOC

The server is refusing to process a request because the request entity 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.

9.4.15. 414 Request-URI Too Long

TOC

The server is refusing to service the request because the 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-URI.

9.4.16. 415 Unsupported Media Type

TOC

The server is refusing to service the request because the entity of the request is in a format not supported by the requested resource for the requested method.

9.4.17. 416 Requested Range Not Satisfiable

TOC

The request included a Range request-header field (Section 6.4 of [Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 5: Range Requests and Partial Responses,” June 2008.)) and none of the range-specifier values in this field overlap the current extent of the selected resource.

9.4.18. 417 Expectation Failed

TOC

The expectation given in an Expect request-header field (see Section 10.2 (Expect)) 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.

9.5. Server Error 5xx

TOC

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 an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included entity to the user. These response codes are applicable to any request method.

9.5.1. 500 Internal Server Error

TOC

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

9.5.2. 501 Not Implemented

TOC

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.

9.5.3. 502 Bad Gateway

TOC

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.

9.5.4. 503 Service Unavailable

TOC

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. If no Retry-After is given, the client SHOULD handle the response as it would for a 500 response.

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

9.5.5. 504 Gateway Timeout

TOC

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

9.5.6. 505 HTTP Version Not Supported

TOC

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

3.1 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," June 2008.), other than with this error message. The response SHOULD contain an entity describing why that version is not supported and what other protocols are supported by that server.

---

10. Header Field Definitions

TOC

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

For entity-header fields, both sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

10.1. Allow

TOC

The Allow response-header field lists the set of methods advertised as supported by the resource identified by the Request-URI. The purpose of this field is strictly to inform the recipient of valid methods associated with the resource. An Allow header field MUST be present in a 405 (Method Not Allowed) response.

```
Allow   = "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 even if it does not understand all the methods specified, since the user agent might have other means of communicating with the origin server.

10.2. Expect

TOC

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

```
Expect       =  "Expect" ":" 1#expectation

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

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

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

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

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

See Section 7.2.3 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” June 2008.) for the use of the 100 (Continue) status.

10.3. From

TOC

The From request-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 [RFC2822] (Resnick, P., “Internet Message Format,” April 2001.):

```
From    = "From" ":" mailbox

mailbox = <mailbox, defined in [RFC2822], 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 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.

10.4. Location

TOC

The Location response-header field is used for the identification of a new resource or to redirect the recipient to a location other than the Request-URI for completion of the request. For 201 (Created) responses, the Location is that of the new resource which was created by the request. For 3xx responses, the location SHOULD indicate the server's preferred URI for automatic redirection to the resource. The field value consists of a single absolute URI.

```
Location       = "Location" ":" absoluteURI [ "#" fragment ]
```

An example is:

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

> Note: The Content-Location header field (Section 6.7 of [Part3] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 3: Message Payload and Content Negotiation," June 2008.)) differs from Location in that the Content-Location identifies the original location of the entity enclosed in the request. It is therefore possible for a response to contain header fields for both Location and Content-Location.

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

*With a 201 Created response, because in this usage the Location header specifies the URL for the entire created resource.

*With a 300 Multiple Choices, since the choice decision is intended to be made on resource characteristics and not fragment characteristics.

*With 305 Use Proxy.

10.5. Max-Forwards

TOC

The Max-Forwards request-header field provides a mechanism with the TRACE (Section 8.8 (TRACE)) and OPTIONS (Section 8.2 (OPTIONS)) methods to limit the number of proxies or gateways that can forward the request to the next inbound server. This can be useful when the client is attempting to trace a request chain which appears to be failing or looping in mid-chain.

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

The Max-Forwards value is a decimal integer indicating the remaining number of times this request message may be forwarded.

Each proxy or gateway 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 methods defined by this specification and for any extension methods for which it is not explicitly referred to as part of that method definition.

10.6. Referer

TOC

The Referer[sic] request-header field allows the client to specify, for the server's benefit, the address (URI) of the resource from which the Request-URI was obtained (the "referrer", although the header field is misspelled.) The Referer request-header allows a server 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. The Referer field MUST NOT be sent if the Request-URI was obtained from a source that does not have its own URI, such as input from the user keyboard.

```
    Referer        = "Referer" ":" ( absoluteURI | relativeURI )
```

Example:

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

If the field value is a relative URI, it SHOULD be interpreted relative to the Request-URI. The URI MUST NOT include a fragment. See Section 12.2 (Encoding Sensitive Information in URIs) for security considerations.

10.7. Retry-After

TOC

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

```
    Retry-After   = "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.

10.8. Server

TOC

The Server response-header field contains information about the software used by the origin server to handle the request. The field can contain multiple product tokens (Section 3.5 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," June 2008.)) and comments identifying the server and any significant subproducts. The product tokens are listed in order of their significance for identifying the application.

```
Server         = "Server" ":" 1*( 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 response-header. Instead, it MUST include a Via field (as described in Section 8.9 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," June 2008.)).

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

10.9. User-Agent

TOC

The User-Agent request-header field contains information about the user agent originating the request. This is for statistical purposes, the tracing of protocol violations, and automated

recognition of user agents for the sake of tailoring responses to avoid particular user agent limitations. User agents SHOULD include this field with requests. The field can contain multiple product tokens (Section 3.5 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," June 2008.)) and comments identifying the agent and any subproducts which form a significant part of the user agent. By convention, the product tokens are listed in order of their significance for identifying the application.

```
    User-Agent     = "User-Agent" ":" 1*( product | comment )
```

Example:

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

11. IANA Considerations

TOC

11.1. Status Code Registry

TOC

The registration procedure for HTTP Status Codes -- previously defined in Section 7.1 of [RFC2817] (Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1," May 2000.) -- is now defined by Section 5.1 (Status Code Registry) of this document.

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

Value	Description	Reference
100	Continue	Section 9.1.1 (100 Continue)
101	Switching Protocols	Section 9.1.2 (101 Switching Protocols)
200	OK	Section 9.2.1 (200 OK)
201	Created	Section 9.2.2 (201 Created)
202	Accepted	Section 9.2.3 (202 Accepted)
203	Non-Authoritative Information	Section 9.2.4 (203 Non-Authoritative Information)
204	No Content	Section 9.2.5 (204 No Content)
205	Reset Content	Section 9.2.6 (205 Reset Content)
206	Partial Content	Section 9.2.7 (206 Partial Content)
300	Multiple Choices	Section 9.3.1 (300 Multiple Choices)
301	Moved Permanently	Section 9.3.2 (301 Moved Permanently)

302	Found	Section 9.3.3 (302 Found)
303	See Other	Section 9.3.4 (303 See Other)
304	Not Modified	Section 9.3.5 (304 Not Modified)
305	Use Proxy	Section 9.3.6 (305 Use Proxy)
306	(Unused)	Section 9.3.7 (306 (Unused))
307	Temporary Redirect	Section 9.3.8 (307 Temporary Redirect)
400	Bad Request	Section 9.4.1 (400 Bad Request)
401	Unauthorized	Section 9.4.2 (401 Unauthorized)
402	Payment Required	Section 9.4.3 (402 Payment Required)
403	Forbidden	Section 9.4.4 (403 Forbidden)
404	Not Found	Section 9.4.5 (404 Not Found)
405	Method Not Allowed	Section 9.4.6 (405 Method Not Allowed)
406	Not Acceptable	Section 9.4.7 (406 Not Acceptable)
407	Proxy Authentication Required	Section 9.4.8 (407 Proxy Authentication Required)
408	Request Timeout	Section 9.4.9 (408 Request Timeout)
409	Conflict	Section 9.4.10 (409 Conflict)
410	Gone	Section 9.4.11 (410 Gone)
411	Length Required	Section 9.4.12 (411 Length Required)
412	Precondition Failed	Section 9.4.13 (412 Precondition Failed)
413	Request Entity Too Large	Section 9.4.14 (413 Request Entity Too Large)
414	Request-URI Too Long	Section 9.4.15 (414 Request-URI Too Long)
415	Unsupported Media Type	Section 9.4.16 (415 Unsupported Media Type)
416	Requested Range Not Satisfiable	Section 9.4.17 (416 Requested Range Not Satisfiable)
417	Expectation Failed	Section 9.4.18 (417 Expectation Failed)
500	Internal Server Error	Section 9.5.1 (500 Internal Server Error)
501	Not Implemented	Section 9.5.2 (501 Not Implemented)
502	Bad Gateway	Section 9.5.3 (502 Bad Gateway)
503	Service Unavailable	Section 9.5.4 (503 Service Unavailable)
504	Gateway Timeout	Section 9.5.5 (504 Gateway Timeout)
505	HTTP Version Not Supported	Section 9.5.6 (505 HTTP Version Not Supported)

11.2. Message Header Registration

TOC

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

(Klyne, G., Nottingham, M., and J. Mogul, “Registration Procedures for Message Header Fields,” September 2004.):

Header Field Name	Protocol	Status	Reference
Allow	http	standard	Section 10.1 (Allow)
Expect	http	standard	Section 10.2 (Expect)
From	http	standard	Section 10.3 (From)
Location	http	standard	Section 10.4 (Location)
Max-Forwards	http	standard	Section 10.5 (Max-Forwards)
Referer	http	standard	Section 10.6 (Referer)
Retry-After	http	standard	Section 10.7 (Retry-After)
Server	http	standard	Section 10.8 (Server)
User-Agent	http	standard	Section 10.9 (User-Agent)

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

---

12. Security Considerations

TOC

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.

12.1. Transfer of Sensitive Information

TOC

Like any generic data transfer protocol, HTTP cannot regulate the content of the data that is transferred, nor is there any a priori method of determining the sensitivity of any particular piece of information within the context of any given request. Therefore, applications SHOULD supply as much control over this information as possible to the provider of that information. Four header fields are worth special mention in this context: Server, Via, Referer and From.

Revealing the specific software version of the server might allow the server machine to become more vulnerable to attacks against software that is known to contain security holes. Implementors SHOULD make the Server header field a configurable option.

Proxies which serve as a portal through a network firewall SHOULD take special precautions regarding the transfer of header information that identifies the hosts behind the firewall. In

particular, they SHOULD remove, or replace with sanitized versions, any Via fields generated behind the firewall.

The Referer header 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 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 10.9 (User-Agent)) or Server (Section 10.8 (Server)) header fields can sometimes be used to determine that a specific client or server have a particular security hole which might be exploited. Unfortunately, this same information is often used for other valuable purposes for which HTTP currently has no better mechanism.

12.2. Encoding Sensitive Information in URIs

TOC

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 encoded in the Request-URI. Many existing servers, proxies, and user agents log or display the Request-URI in places where it might be visible to third parties. Such services can use POST-based form submission instead.

TOC

12.3. Location Headers and Spoofing

If a single server supports multiple organizations that do not trust one another, then it MUST check the values of Location and Content-Location headers 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.

13. Acknowledgments

TOC

14. References

TOC

14.1. Normative References

TOC

[Part1]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 1: URIs, Connections, and Message Parsing," draft-ietf-httpbis-p1-messaging-03 (work in progress), June 2008.
[Part3]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 3: Message Payload and Content Negotiation," draft-ietf-httpbis-p3-payload-03 (work in progress), June 2008.
[Part4]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests," draft-ietf-httpbis-p4-conditional-03 (work in progress), June 2008.
[Part5]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses," draft-ietf-httpbis-p5-range-03 (work in progress), June 2008.
[Part6]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching," draft-ietf-httpbis-p6-cache-03 (work in progress), June 2008.
[Part7]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 7: Authentication," draft-ietf-httpbis-p7-auth-03 (work in progress), June 2008.
[RFC2119]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997.

14.2. Informative References

TOC

[Luo1998]	Luotonen, A., "Tunneling TCP based protocols through Web proxy servers," draft-luotonen-web-proxy-tunneling-01 (work in progress), August 1998.
[RFC1945]	Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0," RFC 1945, May 1996.
[RFC2068]	Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2068, January 1997.
[RFC2616]	Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616, June 1999.
[RFC2817]	Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1," RFC 2817, May 2000.
[RFC2822]	Resnick, P., "Internet Message Format," RFC 2822, April 2001.
[RFC3864]	Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields," BCP 90, RFC 3864, September 2004.
[RFC5226]	Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," BCP 26, RFC 5226, May 2008.

Appendix A. Compatibility with Previous Versions

TOC

A.1. Changes from RFC 2068

TOC

Clarified which error code should be used for inbound server failures (e.g. DNS failures). (Section 9.5.5 (504 Gateway Timeout)).

201 (Created) had a race that required an Etag be sent when a resource is first created. (Section 9.2.2 (201 Created)).

Rewrite of message transmission requirements to make it much harder for implementors to get it wrong, as the consequences of errors here can have significant impact on the Internet, and to deal with the following problems:

1. Changing "HTTP/1.1 or later" to "HTTP/1.1", in contexts where this was incorrectly placing a requirement on the behavior of an implementation of a future version of HTTP/1.x

2. Made it clear that user-agents should retry requests, not "clients" in general.

3. Converted requirements for clients to ignore unexpected 100 (Continue) responses, and for proxies to forward 100 responses, into a general requirement for 1xx responses.

4. Modified some TCP-specific language, to make it clearer that non-TCP transports are possible for HTTP.

5. Require that the origin server MUST NOT wait for the request body before it sends a required 100 (Continue) response.

6. Allow, rather than require, a server to omit 100 (Continue) if it has already seen some of the request body.

7. Allow servers to defend against denial-of-service attacks and broken clients.

This change adds the Expect header and 417 status code.

Clean up confusion between 403 and 404 responses. (Section 9.4.4 (403 Forbidden), 9.4.5 (404 Not Found), and 9.4.11 (410 Gone))

The PATCH, LINK, UNLINK methods were defined but not commonly implemented in previous versions of this specification. See Section 19.6.1 of [RFC2068] (Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," January 1997.).

A.2. Changes from RFC 2616

TOC

This document takes over the Status Code Registry, previously defined in Section 7.1 of [RFC2817] (Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1," May 2000.). (Section 5.1 (Status Code Registry))

Clarify definition of POST. (Section 8.5 (POST))

Failed to consider that there are many other request methods that are safe to automatically redirect, and further that the user agent is able to make that determination based on the request method semantics. (Sections 9.3.2 (301 Moved Permanently), 9.3.3 (302 Found) and 9.3.8 (307 Temporary Redirect))

Deprecate 305 Use Proxy status code, because user agents did not implement it. It used to indicate that the requested resource must be accessed through the proxy given by the Location field. The Location field gave the URI of the proxy. The recipient was expected to repeat this single request via the proxy. (Section 9.3.6 (305 Use Proxy))

Reclassify Allow header as response header, removing the option to specify it in a PUT request. Relax the server requirement on the contents of the Allow header and remove requirement on clients to always trust the header value. (Section 10.1 (Allow))

Correct syntax of Location header to allow fragment, as referred symbol wasn't what was expected, and add some clarifications as to when it would not be appropriate. (Section 10.4 (Location))

In the description of the Server header, the Via field was described as a SHOULD. The requirement was and is stated correctly in the description of the Via header in Section 8.9 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” June 2008.). (Section 10.8 (Server))

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

TOC

B.1. Since RFC2616

TOC

Extracted relevant partitions from [RFC2616] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.).

B.2. Since draft-ietf-httpbis-p2-semantics-00

TOC

Closed issues:

- http://www3.tools.ietf.org/wg/httpbis/trac/ticket/5: "Via is a MUST" (http://purl.org/NET/http-errata#via-must)
- http://www3.tools.ietf.org/wg/httpbis/trac/ticket/6: "Fragments allowed in Location" (http://purl.org/NET/http-errata#location-fragments)
- http://www3.tools.ietf.org/wg/httpbis/trac/ticket/10: "Safe Methods vs Redirection" (http://purl.org/NET/http-errata#saferedirect)
- http://www3.tools.ietf.org/wg/httpbis/trac/ticket/17: "Revise description of the POST method" (http://purl.org/NET/http-errata#post)
- http://www3.tools.ietf.org/wg/httpbis/trac/ticket/35: "Normative and Informative references"
- http://www3.tools.ietf.org/wg/httpbis/trac/ticket/42: "RFC2606 Compliance"
- http://www3.tools.ietf.org/wg/httpbis/trac/ticket/65: "Informative references"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/84: "Redundant cross-references"

Other changes:

* Move definitions of 304 and 412 condition codes to [Part4] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 4: Conditional Requests," June 2008.)

B.3. Since draft-ietf-httpbis-p2-semantics-01

TOC

Closed issues:

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/21: "PUT side effects"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/91: "Duplicate Host header requirements"

Ongoing work on ABNF conversion (http://www3.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.

* 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.

B.4. Since draft-ietf-httpbis-p2-semantics-02

TOC

Closed issues:

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/24: "Requiring Allow in 405 responses"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/59: "Status Code Registry"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/61: "Redirection vs. Location"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/70: "Cacheability of 303 response"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/76: "305 Use Proxy"

*http://www3.tools.ietf.org/wg/httpbis/trac/ticket/105: "Classification for Allow header"

*http://www3.tools.ietf.org/wg/httpbis/trac/ticket/112: "PUT - 'store under' vs 'store at'"

Ongoing work on IANA Message Header Registration (http://www3.tools.ietf.org/wg/httpbis/trac/ticket/40):

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

Ongoing work on ABNF conversion (http://www3.tools.ietf.org/wg/httpbis/trac/ticket/36):

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

Index

TOC

Authors' Addresses

TOC

	Roy T. Fielding (editor)
	Day Software
	23 Corporate Plaza DR, Suite 280
	Newport Beach, CA 92660
	USA
Phone:	+1-949-706-5300
Fax:	+1-949-706-5305
Email:	fielding@gbiv.com
URI:	http://roy.gbiv.com/
	Jim Gettys
	One Laptop per Child
	21 Oak Knoll Road
	Carlisle, MA 01741
	USA
Email:	jg@laptop.org
URI:	http://www.laptop.org/
	Jeffrey C. Mogul
	Hewlett-Packard Company
	HP Labs, Large Scale Systems Group
	1501 Page Mill Road, MS 1177
	Palo Alto, CA 94304
	USA
Email:	JeffMogul@acm.org

	Henrik Frystyk Nielsen
	Microsoft Corporation
	1 Microsoft Way
	Redmond, WA 98052
	USA
Email:	henrikn@microsoft.com
	Larry Masinter
	Adobe Systems, Incorporated
	345 Park Ave
	San Jose, CA 95110
	USA
Email:	LMM@acm.org
URI:	http://larry.masinter.net/
	Paul J. Leach
	Microsoft Corporation
	1 Microsoft Way
	Redmond, WA 98052
Email:	paulle@microsoft.com
	Tim Berners-Lee
	World Wide Web Consortium
	MIT Computer Science and Artificial Intelligence Laboratory
	The Stata Center, Building 32
	32 Vassar Street
	Cambridge, MA 02139
	USA
Email:	timbl@w3.org
URI:	http://www.w3.org/People/Berners-Lee/
	Yves Lafon (editor)
	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
Phone:	+49 251 2807760
Fax:	+49 251 2807761
Email:	julian.reschke@greenbytes.de
URI:	http://greenbytes.de/tech/webdav/

Full Copyright Statement

TOC

Intellectual Property