

MUST generate an Upgrade header field in the response that indicates which protocol(s) will be switched to immediately after the empty line that terminates the 101 response.

It is assumed that the server will only agree to switch protocols when it is advantageous to do so. For example, switching to a newer version of HTTP might be advantageous over older versions, and switching to a real-time, synchronous protocol might be advantageous when delivering resources that use such features.

6.3. Successful 2xx

The 2xx (Successful) class of status code indicates that the client's request was successfully received, understood, and accepted.

6.3.1. 200 OK

The 200 (OK) status code indicates that the request has succeeded. The payload sent in a 200 response depends on the request method. For the methods defined by this specification, the intended meaning of the payload can be summarized as:

GET a representation of the target resource;

HEAD the same representation as GET, but without the representation data;

POST a representation of the status of, or results obtained from, the action;

PUT, DELETE a representation of the status of the action;

OPTIONS a representation of the communications options;

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

Aside from responses to CONNECT, a 200 response always has a payload, though an origin server MAY generate a payload body of zero length. If no payload is desired, an origin server ought to send 204 (No Content) instead. For CONNECT, no payload is allowed because the successful result is a tunnel, which begins immediately after the 200 response header section.

A 200 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.3.2. 201 Created

The 201 (Created) status code indicates that the request has been fulfilled and has resulted in one or more new resources being created. The primary resource created by the request is identified by either a Location header field in the response or, if no Location field is received, by the effective request URI.

The 201 response payload typically describes and links to the resource(s) created. See Section 7.2 for a discussion of the meaning and purpose of validator header fields, such as ETag and Last-Modified, in a 201 response.

6.3.3. 202 Accepted

The 202 (Accepted) status code indicates that 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 in HTTP for re-sending a status code from an asynchronous operation.

The 202 response is intentionally noncommittal. 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 sent with this response ought to describe the request's current status and point to (or embed) a status monitor that can provide the user with an estimate of when the request will be fulfilled.

6.3.4. 203 Non-Authoritative Information

The 203 (Non-Authoritative Information) status code indicates that the request was successful but the enclosed payload has been modified from that of the origin server's 200 (OK) response by a transforming proxy (Section 5.7.2 of [RFC7230]). This status code allows the proxy to notify recipients when a transformation has been applied, since that knowledge might impact later decisions regarding the content. For example, future cache validation requests for the content might only be applicable along the same request path (through the same proxies).

The 203 response is similar to the Warning code of 214 Transformation Applied (Section 5.5 of [RFC7234]), which has the advantage of being applicable to responses with any status code.

A 203 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.3.5. 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 send in the response payload body. Metadata in the response header fields refer to the target resource and its selected representation after the requested action was applied.

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 does not need to 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 its 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.

A 204 response is terminated by the first empty line after the header fields because it cannot contain a message body.

A 204 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.3.6. 205 Reset Content

The 205 (Reset Content) status code indicates that the server has fulfilled the request and desires that the user agent reset the "document view", which caused the request to be sent, to its original state as received from the origin server.

This response is intended to support a common data entry use case where the user receives content that supports data entry (a form, notepad, canvas, etc.), enters or manipulates data in that space,

causes the entered data to be submitted in a request, and then the data entry mechanism is reset for the next entry so that the user can easily initiate another input action.

Since the 205 status code implies that no additional content will be provided, a server **MUST NOT** generate a payload in a 205 response. In other words, a server **MUST** do one of the following for a 205 response: a) indicate a zero-length body for the response by including a Content-Length header field with a value of 0; b) indicate a zero-length payload for the response by including a Transfer-Encoding header field with a value of chunked and a message body consisting of a single chunk of zero-length; or, c) close the connection immediately after sending the blank line terminating the header section.

6.4. Redirection 3xx

The 3xx (Redirection) class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. If a Location header field (Section 7.1.2) is provided, the user agent **MAY** automatically redirect its request to the URI referenced by the Location field value, even if the specific status code is not understood. Automatic redirection needs to done with care for methods not known to be safe, as defined in Section 4.2.1, since the user might not wish to redirect an unsafe request.

There are several types of redirects:

1. Redirects that indicate the resource might be available at a different URI, as provided by the Location field, as in the status codes 301 (Moved Permanently), 302 (Found), and 307 (Temporary Redirect).
2. Redirection that offers a choice of matching resources, each capable of representing the original request target, as in the 300 (Multiple Choices) status code.
3. Redirection to a different resource, identified by the Location field, that can represent an indirect response to the request, as in the 303 (See Other) status code.
4. Redirection to a previously cached result, as in the 304 (Not Modified) status code.

Note: In HTTP/1.0, the status codes 301 (Moved Permanently) and 302 (Found) were defined for the first type of redirect ([RFC1945], Section 9.3). Early user agents split on whether the method applied to the redirect target would be the same as the

original request or would be rewritten as GET. Although HTTP originally defined the former semantics for 301 and 302 (to match its original implementation at CERN), and defined 303 (See Other) to match the latter semantics, prevailing practice gradually converged on the latter semantics for 301 and 302 as well. The first revision of HTTP/1.1 added 307 (Temporary Redirect) to indicate the former semantics without being impacted by divergent practice. Over 10 years later, most user agents still do method rewriting for 301 and 302; therefore, this specification makes that behavior conformant when the original request is POST.

A client 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.

6.4.1. 300 Multiple Choices

The 300 (Multiple Choices) status code indicates that the target resource has more than one representation, each with its own more specific identifier, and information about the alternatives is being provided so that the user (or user agent) can select a preferred representation by redirecting its request to one or more of those identifiers. In other words, the server desires that the user agent engage in reactive negotiation to select the most appropriate representation(s) for its needs (Section 3.4).

If the server has a preferred choice, the server SHOULD generate a Location header field containing a preferred choice's URI reference. The user agent MAY use the Location field value for automatic redirection.

For request methods other than HEAD, the server SHOULD generate a payload in the 300 response containing a list of representation metadata and URI reference(s) from which the user or user agent can choose the one most preferred. The user agent MAY make a selection from that list automatically if it understands the provided media type. A specific format for automatic selection is not defined by this specification because HTTP tries to remain orthogonal to the definition of its payloads. In practice, the representation is provided in some easily parsed format believed to be acceptable to the user agent, as determined by shared design or content negotiation, or in some commonly accepted hypertext format.

A 300 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

Note: The original proposal for the 300 status code defined the URI header field as providing a list of alternative representations, such that it would be usable for 200, 300, and 406 responses and be transferred in responses to the HEAD method. However, lack of deployment and disagreement over syntax led to both URI and Alternates (a subsequent proposal) being dropped from this specification. It is possible to communicate the list using a set of Link header fields [RFC5988], each with a relationship of "alternate", though deployment is a chicken-and-egg problem.

6.4.2. 301 Moved Permanently

The 301 (Moved Permanently) status code indicates that the target resource has been assigned a new permanent URI and any future references to this resource ought to use one of the enclosed 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 sent by the server, where possible.

The server SHOULD generate a Location header field in the response containing a preferred URI reference for the new permanent URI. The user agent MAY use the Location field value for automatic redirection. The server's response payload usually contains a short hypertext note with a hyperlink to the new URI(s).

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

A 301 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.4.3. 302 Found

The 302 (Found) status code indicates that the target resource resides temporarily under a different URI. Since the redirection might be altered on occasion, the client ought to continue to use the effective request URI for future requests.

The server SHOULD generate a Location header field in the response containing a URI reference for the different URI. The user agent MAY use the Location field value for automatic redirection. The server's response payload usually contains a short hypertext note with a hyperlink to the different URI(s).

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

6.4.4. 303 See Other

The 303 (See Other) 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, which is intended to provide an indirect response to the original request. A user agent can perform a retrieval request targeting that URI (a GET or HEAD request if using HTTP), which might also be redirected, 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 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 origin server does not have a representation of the target resource that can be transferred by the server over HTTP. However, the Location field value refers to a resource that is descriptive of the target resource, such that making a retrieval request on that other resource might result in a representation that is useful to recipients without implying that it represents the original 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.

Except for responses to a HEAD request, the representation of a 303 response ought to contain a short hypertext note with a hyperlink to the same URI reference provided in the Location header field.

6.4.5. 305 Use Proxy

The 305 (Use Proxy) status code was defined in a previous version of this specification and is now deprecated (Appendix B).

6.4.6. 306 (Unused)

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

6.4.7. 307 Temporary Redirect

The 307 (Temporary Redirect) status code indicates that the target resource resides temporarily under a different URI and the user agent MUST NOT change the request method if it performs an automatic redirection to that URI. Since the redirection can change over time, the client ought to continue using the original effective request URI for future requests.

The server SHOULD generate a Location header field in the response containing a URI reference for the different URI. The user agent MAY use the Location field value for automatic redirection. The server's response payload usually contains a short hypertext note with a hyperlink to the different URI(s).

Note: This status code is similar to 302 (Found), except that it does not allow changing the request method from POST to GET. This specification defines no equivalent counterpart for 301 (Moved Permanently) ([RFC7238], however, defines the status code 308 (Permanent Redirect) for this purpose).

6.5. Client Error 4xx

The 4xx (Client Error) class of status code indicates that the client seems to have erred. Except when responding to a HEAD request, the server SHOULD send 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.

6.5.1. 400 Bad Request

The 400 (Bad Request) status code indicates that the server cannot or will not process the request due to something that is perceived to be a client error (e.g., malformed request syntax, invalid request message framing, or deceptive request routing).

6.5.2. 402 Payment Required

The 402 (Payment Required) status code is reserved for future use.

6.5.3. 403 Forbidden

The 403 (Forbidden) status code indicates that the server understood the request but refuses to authorize it. A server that wishes to make public why the request has been forbidden can describe that reason in the response payload (if any).

If authentication credentials were provided in the request, the server considers them insufficient to grant access. The client SHOULD NOT automatically repeat the request with the same credentials. The client MAY repeat the request with new or different credentials. However, a request might be forbidden for reasons unrelated to the credentials.

An origin server that wishes to "hide" the current existence of a forbidden target resource MAY instead respond with a status code of 404 (Not Found).

6.5.4. 404 Not Found

The 404 (Not Found) status code indicates that the origin server did not find a current representation for the target resource or is not willing to disclose that one exists. A 404 status code does not indicate whether this lack of representation is temporary or permanent; the 410 (Gone) status code is preferred over 404 if the origin server knows, presumably through some configurable means, that the condition is likely to be permanent.

A 404 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.5.5. 405 Method Not Allowed

The 405 (Method Not Allowed) status code indicates that the method received in the request-line is known by the origin server but not supported by the target resource. The origin server MUST generate an Allow header field in a 405 response containing a list of the target resource's currently supported methods.

A 405 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.5.6. 406 Not Acceptable

The 406 (Not Acceptable) status code indicates that the target resource does not have a current representation that would be acceptable to the user agent, according to the proactive negotiation header fields received in the request (Section 5.3), and the server is unwilling to supply a default representation.

The server SHOULD generate a payload containing a list of available representation characteristics and corresponding resource identifiers from which the user or user agent can choose the one most appropriate. A user agent MAY automatically select the most appropriate choice from that list. However, this specification does not define any standard for such automatic selection, as described in Section 6.4.1.

6.5.7. 408 Request Timeout

The 408 (Request Timeout) status code indicates that the server did not receive a complete request message within the time that it was prepared to wait. A server SHOULD send the "close" connection option (Section 6.1 of [RFC7230]) in the response, since 408 implies that the server has decided to close the connection rather than continue waiting. If the client has an outstanding request in transit, the client MAY repeat that request on a new connection.

6.5.8. 409 Conflict

The 409 (Conflict) status code indicates that the request could not be completed due to a conflict with the current state of the target resource. This code is used in situations where the user might be able to resolve the conflict and resubmit the request. The server SHOULD generate a payload that includes enough information for a user to recognize the source of the conflict.

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 that conflict with those made by an earlier (third-party) request, the origin server might use a 409 response to indicate that it can't complete the request. In this case, the response representation would likely contain information useful for merging the differences based on the revision history.

6.5.9. 410 Gone

The 410 (Gone) status code indicates that access to the target resource is no longer available at the origin server and that this condition is likely to be permanent. If the origin server does not

know, or has no facility to determine, whether or not the condition is permanent, the status code 404 (Not Found) ought to 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 associated with the origin 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.

A 410 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.5.10. 411 Length Required

The 411 (Length Required) status code indicates that the server refuses to accept the request without a defined Content-Length (Section 3.3.2 of [RFC7230]). 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.

6.5.11. 413 Payload Too Large

The 413 (Payload Too Large) status code indicates that the server is refusing to process a request because the request payload 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 generate a Retry-After header field to indicate that it is temporary and after what time the client MAY try again.

6.5.12. 414 URI Too Long

The 414 (URI Too Long) status code indicates that the server is refusing to service the request because the request-target (Section 5.3 of [RFC7230]) 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 "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 potential security holes.

A 414 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.5.13. 415 Unsupported Media Type

The 415 (Unsupported Media Type) status code indicates that the origin server is refusing to service the request because the payload is in a format not supported by this method on the target resource. The format problem might be due to the request's indicated Content-Type or Content-Encoding, or as a result of inspecting the data directly.

6.5.14. 417 Expectation Failed

The 417 (Expectation Failed) status code indicates that the expectation given in the request's Expect header field (Section 5.1.1) could not be met by at least one of the inbound servers.

6.5.15. 426 Upgrade Required

The 426 (Upgrade Required) status code indicates that the server refuses to perform the request using the current protocol but might be willing to do so after the client upgrades to a different protocol. The server MUST send an Upgrade header field in a 426 response to indicate the required protocol(s) (Section 6.7 of [RFC7230]).

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.

6.6. Server Error 5xx

The 5xx (Server Error) class of status code indicates that the server is aware that it has erred or is incapable of performing the requested method. Except when responding to a HEAD request, the server SHOULD send a representation containing an explanation of the error situation, and whether it is a temporary or permanent

condition. A user agent SHOULD display any included representation to the user. These response codes are applicable to any request method.

6.6.1. 500 Internal Server Error

The 500 (Internal Server Error) status code indicates that the server encountered an unexpected condition that prevented it from fulfilling the request.

6.6.2. 501 Not Implemented

The 501 (Not Implemented) status code indicates that 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.

A 501 response is cacheable by default; i.e., unless otherwise indicated by the method definition or explicit cache controls (see Section 4.2.2 of [RFC7234]).

6.6.3. 502 Bad Gateway

The 502 (Bad Gateway) status code indicates that the server, while acting as a gateway or proxy, received an invalid response from an inbound server it accessed while attempting to fulfill the request.

6.6.4. 503 Service Unavailable

The 503 (Service Unavailable) status code indicates that the server is currently unable to handle the request due to a temporary overload or scheduled maintenance, which will likely be alleviated after some delay. The server MAY send a Retry-After header field (Section 7.1.3) to suggest an appropriate amount of time for the client to wait before retrying the request.

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

6.6.5. 504 Gateway Timeout

The 504 (Gateway Timeout) status code indicates that the server, while acting as a gateway or proxy, did not receive a timely response from an upstream server it needed to access in order to complete the request.

6.6.6. 505 HTTP Version Not Supported

The 505 (HTTP Version Not Supported) status code indicates that the server does not support, or refuses to support, the major version of HTTP 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 2.6 of [RFC7230], other than with this error message. The server **SHOULD** generate a representation for the 505 response that describes why that version is not supported and what other protocols are supported by that server.

7. Response Header Fields

The response header fields allow the server to pass additional information about the response beyond what is placed in the status-line. These header fields give information about the server, about further access to the target resource, or about related resources.

Although each response header field has a defined meaning, in general, the precise semantics might be further refined by the semantics of the request method and/or response status code.

7.1. Control Data

Response header fields can supply control data that supplements the status code, directs caching, or instructs the client where to go next.

Header Field Name	Defined in...
Age	Section 5.1 of [RFC7234]
Cache-Control	Section 5.2 of [RFC7234]
Expires	Section 5.3 of [RFC7234]
Date	Section 7.1.1.2
Location	Section 7.1.2
Retry-After	Section 7.1.3
Vary	Section 7.1.4
Warning	Section 5.5 of [RFC7234]

7.1.1. Origination Date

7.1.1.1. Date/Time Formats

Prior to 1995, there were three different formats commonly used by servers to communicate timestamps. For compatibility with old implementations, all three are defined here. The preferred format is a fixed-length and single-zone subset of the date and time specification used by the Internet Message Format [RFC5322].

HTTP-date = IMF-fixdate / obs-date

An example of the preferred format is

Sun, 06 Nov 1994 08:49:37 GMT ; IMF-fixdate

Examples of the two obsolete formats are

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

A recipient that parses a timestamp value in an HTTP header field MUST accept all three HTTP-date formats. When a sender generates a header field that contains one or more timestamps defined as HTTP-date, the sender MUST generate those timestamps in the IMF-fixdate format.

An HTTP-date value represents time as an instance of Coordinated Universal Time (UTC). The first two formats indicate UTC by the three-letter abbreviation for Greenwich Mean Time, "GMT", a predecessor of the UTC name; values in the asctime format are assumed to be in UTC. A sender that generates HTTP-date values from a local clock ought to use NTP ([RFC5905]) or some similar protocol to synchronize its clock to UTC.

Preferred format:

IMF-fixdate = day-name "," SP date1 SP time-of-day SP GMT
 ; fixed length/zone/capitalization subset of the format
 ; see Section 3.3 of [RFC5322]

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 - 23:59:60 (leap second)

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

```

Obsolete formats:

```

obs-date   = rfc850-date / asctime-date

rfc850-date = day-name-1 "," SP date2 SP time-of-day SP GMT
date2      = day "-" month "-" 2DIGIT
            ; e.g., 02-Jun-82

day-name-1 = %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 ) )
            ; e.g., Jun 2

```

HTTP-date is case sensitive. A sender MUST NOT generate additional whitespace in an HTTP-date beyond that specifically included as SP in the grammar. The semantics of day-name, day, month, year, and time-of-day are the same as those defined for the Internet Message Format constructs with the corresponding name ([RFC5322], Section 3.3).

Recipients of a timestamp value in rfc850-date format, which uses a two-digit year, MUST interpret a timestamp that appears to be more than 50 years in the future as representing the most recent year in the past that had the same last two digits.

Recipients of timestamp values are encouraged to be robust in parsing timestamps unless otherwise restricted by the field definition. For example, messages are occasionally forwarded over HTTP from a non-HTTP source that might generate any of the date and time specifications defined by the Internet Message Format.

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

7.1.1.2. 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 7.1.1.1.

Date = HTTP-date

An example is

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

When a Date header field is generated, the sender SHOULD generate its field value as the best available approximation of the date and time of message generation. 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 message origination.

An origin server MUST NOT send a Date header field if it does not have a clock capable of providing a reasonable approximation of the current instance in Coordinated Universal Time. An origin server MAY send a Date header field if the response is in the 1xx (Informational) or 5xx (Server Error) class of status codes. An origin server MUST send a Date header field in all other cases.

A recipient with a clock that receives a response message without a Date header field MUST record the time it was received and append a corresponding Date header field to the message's header section if it is cached or forwarded downstream.

A user agent MAY send a Date header field in a request, though generally will not do so unless it is believed to convey useful information to the server. For example, custom applications of HTTP might convey a Date if the server is expected to adjust its interpretation of the user's request based on differences between the user agent and server clocks.

7.1.2. Location

The "Location" header field is used in some responses to refer to a specific resource in relation to the response. The type of relationship is defined by the combination of request method and status code semantics.

Location = URI-reference

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).

For 201 (Created) responses, the Location value refers to the primary resource created by the request. For 3xx (Redirection) responses, the Location value refers to the preferred target resource for automatically redirecting the request.

If the Location value provided in a 3xx (Redirection) response does not have a fragment component, a user agent MUST process the redirection as if the value inherits the fragment component of the URI reference used to generate the request target (i.e., the redirection inherits the original reference's fragment, if any).

For example, a GET request generated for the URI reference "http://www.example.org/~tim" might result in a 303 (See Other) response containing the header field:

Location: /People.html#tim

which suggests that the user agent redirect to "http://www.example.org/People.html#tim"

Likewise, a GET request generated for the URI reference "http://www.example.org/index.html#larry" might result in a 301 (Moved Permanently) response containing the header field:

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

which suggests that the user agent redirect to "http://www.example.net/index.html#larry", preserving the original fragment identifier.

There are circumstances in which a fragment identifier in a Location value would not be appropriate. For example, the Location header field in a 201 (Created) response is supposed to provide a URI that is specific to the created resource.

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 for the sake of robustness.

Note: The Content-Location header field (Section 3.1.4.2) differs from Location in that the Content-Location refers to the most specific resource corresponding to the enclosed representation. It is therefore possible for a response to contain both the Location and Content-Location header fields.

7.1.3. Retry-After

Servers send the "Retry-After" header field to indicate how long the user agent ought to wait before making a follow-up request. When sent with a 503 (Service Unavailable) response, Retry-After indicates how long the service is expected to be unavailable to the client. When sent with any 3xx (Redirection) response, Retry-After indicates the minimum time that the user agent is asked to wait before issuing the redirected request.

The value of this field can be either an HTTP-date or a number of seconds to delay after the response is received.

Retry-After = HTTP-date / delay-seconds

A delay-seconds value is a non-negative decimal integer, representing time in seconds.

delay-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.

7.1.4. Vary

The "Vary" header field in a response describes what parts of a request message, aside from the method, Host header field, and request target, might influence the origin server's process for selecting and representing this response. The value consists of either a single asterisk ("*") or a list of header field names (case-insensitive).

```
Vary = "*" / 1#field-name
```

A Vary field value of "*" signals that anything about the request might play a role in selecting the response representation, possibly including elements outside the message syntax (e.g., the client's network address). A recipient will not be able to determine whether this response is appropriate for a later request without forwarding the request to the origin server. A proxy **MUST NOT** generate a Vary field with a "*" value.

A Vary field value consisting of a comma-separated list of names indicates that the named request header fields, known as the selecting header fields, might have a role in selecting the representation. The potential selecting header fields are not limited to those defined by this specification.

For example, a response that contains

```
Vary: accept-encoding, accept-language
```

indicates that the origin server might have used the request's Accept-Encoding and Accept-Language fields (or lack thereof) as determining factors while choosing the content for this response.

An origin server might send Vary with a list of fields for two purposes:

1. To inform cache recipients that they **MUST NOT** use this response to satisfy a later request unless the later request has the same values for the listed fields as the original request (Section 4.1 of [RFC7234]). In other words, Vary expands the cache key required to match a new request to the stored cache entry.

2. To inform user agent recipients that this response is subject to content negotiation (Section 5.3) and that a different representation might be sent in a subsequent request if additional parameters are provided in the listed header fields (proactive negotiation).

An origin server SHOULD send a Vary header field when its algorithm for selecting a representation varies based on aspects of the request message other than the method and request target, unless the variance cannot be crossed or the origin server has been deliberately configured to prevent cache transparency. For example, there is no need to send the Authorization field name in Vary because reuse across users is constrained by the field definition (Section 4.2 of [RFC7235]). Likewise, an origin server might use Cache-Control directives (Section 5.2 of [RFC7234]) to supplant Vary if it considers the variance less significant than the performance cost of Vary's impact on caching.

7.2. Validator Header Fields

Validator header fields convey metadata about the selected representation (Section 3). In responses to safe requests, validator fields describe the selected representation chosen by the origin server while handling the response. Note that, depending on the status code semantics, the selected representation for a given response is not necessarily the same as the representation enclosed as response payload.

In a successful response to a state-changing request, validator fields describe the new representation that has replaced the prior selected representation as a result of processing the request.

For example, an ETag header field in a 201 (Created) response communicates the entity-tag of the newly created resource's representation, so that it can be used in later conditional requests to prevent the "lost update" problem [RFC7232].

Header Field Name	Defined in...
ETag	Section 2.3 of [RFC7232]
Last-Modified	Section 2.2 of [RFC7232]

7.3. Authentication Challenges

Authentication challenges indicate what mechanisms are available for the client to provide authentication credentials in future requests.

Header Field Name	Defined in...
WWW-Authenticate	Section 4.1 of [RFC7235]
Proxy-Authenticate	Section 4.3 of [RFC7235]

7.4. Response Context

The remaining response header fields provide more information about the target resource for potential use in later requests.

Header Field Name	Defined in...
Accept-Ranges	Section 2.3 of [RFC7233]
Allow	Section 7.4.1
Server	Section 7.4.2

7.4.1. Allow

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

Allow = #method

Example of use:

Allow: GET, HEAD, PUT

The actual set of allowed methods is defined by the origin server at the time of each request. An origin server **MUST** generate an Allow field in a 405 (Method Not Allowed) response and **MAY** do so in any other response. An empty Allow field value indicates that the resource allows no methods, which might occur in a 405 response if the resource has been temporarily disabled by configuration.

A proxy **MUST NOT** modify the Allow header field -- it does not need to understand all of the indicated methods in order to handle them according to the generic message handling rules.

7.4.2. Server

The "Server" header field contains information about the software used by the origin server to handle the request, which is often used by clients to help identify the scope of reported interoperability problems, to work around or tailor requests to avoid particular server limitations, and for analytics regarding server or operating system use. An origin server MAY generate a Server field in its responses.

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

The Server field-value consists of one or more product identifiers, each followed by zero or more comments (Section 3.2 of [RFC7230]), which together identify the origin server software and its significant subproducts. By convention, the product identifiers are listed in decreasing order of their significance for identifying the origin server software. Each product identifier consists of a name and optional version, as defined in Section 5.5.3.

Example:

Server: CERN/3.0 libwww/2.17

An origin server SHOULD NOT generate a Server field containing needlessly fine-grained detail and SHOULD limit the addition of subproducts by third parties. Overly long and detailed Server field values increase response latency and potentially reveal internal implementation details that might make it (slightly) easier for attackers to find and exploit known security holes.

8. IANA Considerations

8.1. Method Registry

The "Hypertext Transfer Protocol (HTTP) Method Registry" defines the namespace for the request method token (Section 4). The method registry has been created and is now maintained at <http://www.iana.org/assignments/http-methods>.

8.1.1. Procedure

HTTP method registrations MUST include the following fields:

- o Method Name (see Section 4)
- o Safe ("yes" or "no", see Section 4.2.1)
- o Idempotent ("yes" or "no", see Section 4.2.2)
- o Pointer to specification text

Values to be added to this namespace require IETF Review (see [RFC5226], Section 4.1).

8.1.2. Considerations for New Methods

Standardized methods are generic; that is, they are potentially applicable to any resource, not just one particular media type, kind of resource, or application. As such, it is preferred that new methods be registered in a document that isn't specific to a single application or data format, since orthogonal technologies deserve orthogonal specification.

Since message parsing (Section 3.3 of [RFC7230]) needs to be independent of method semantics (aside from responses to HEAD), definitions of new methods cannot change the parsing algorithm or prohibit the presence of a message body on either the request or the response message. Definitions of new methods can specify that only a zero-length message body is allowed by requiring a Content-Length header field with a value of "0".

A new method definition needs to indicate whether it is safe (Section 4.2.1), idempotent (Section 4.2.2), cacheable (Section 4.2.3), what semantics are to be associated with the payload body if any is present in the request and what refinements the method makes to header field or status code semantics. If the new method is cacheable, its definition ought to describe how, and under what conditions, a cache can store a response and use it to satisfy a subsequent request. The new method ought to describe whether it can be made conditional (Section 5.2) and, if so, how a server responds when the condition is false. Likewise, if the new method might have some use for partial response semantics ([RFC7233]), it ought to document this, too.

Note: Avoid defining a method name that starts with "M-", since that prefix might be misinterpreted as having the semantics assigned to it by [RFC2774].

8.1.3. Registrations

The "Hypertext Transfer Protocol (HTTP) Method Registry" has been populated with the registrations below:

Method	Safe	Idempotent	Reference
CONNECT	no	no	Section 4.3.6
DELETE	no	yes	Section 4.3.5
GET	yes	yes	Section 4.3.1
HEAD	yes	yes	Section 4.3.2
OPTIONS	yes	yes	Section 4.3.7
POST	no	no	Section 4.3.3
PUT	no	yes	Section 4.3.4
TRACE	yes	yes	Section 4.3.8

8.2. Status Code Registry

The "Hypertext Transfer Protocol (HTTP) Status Code Registry" defines the namespace for the response status-code token (Section 6). The status code registry is maintained at <http://www.iana.org/assignments/http-status-codes>.

This section replaces the registration procedure for HTTP Status Codes previously defined in Section 7.1 of [RFC2817].

8.2.1. Procedure

A registration MUST include the following fields:

- o Status Code (3 digits)
- o Short Description
- o Pointer to specification text

Values to be added to the HTTP status code namespace require IETF Review (see [RFC5226], Section 4.1).

8.2.2. Considerations for New Status Codes

When it is necessary to express semantics for a response that are not defined by current status codes, a new status code can be registered. Status codes are generic; they are potentially applicable to any resource, not just one particular media type, kind of resource, or application of HTTP. As such, it is preferred that new status codes be registered in a document that isn't specific to a single application.

New status codes are required to fall under one of the categories defined in Section 6. To allow existing parsers to process the response message, new status codes cannot disallow a payload, although they can mandate a zero-length payload body.

Proposals for new status codes that are not yet widely deployed ought to avoid allocating a specific number for the code until there is clear consensus that it will be registered; instead, early drafts can use a notation such as "4NN", or "3N0" .. "3N9", to indicate the class of the proposed status code(s) without consuming a number prematurely.

The definition of a new status code ought to explain the request conditions that would cause a response containing that status code (e.g., combinations of request header fields and/or method(s)) along with any dependencies on response header fields (e.g., what fields are required, what fields can modify the semantics, and what header field semantics are further refined when used with the new status code).

The definition of a new status code ought to specify whether or not it is cacheable. Note that all status codes can be cached if the response they occur in has explicit freshness information; however, status codes that are defined as being cacheable are allowed to be cached without explicit freshness information. Likewise, the definition of a status code can place constraints upon cache behavior. See [RFC7234] for more information.

Finally, the definition of a new status code ought to indicate whether the payload has any implied association with an identified resource (Section 3.1.4.1).

8.2.3. Registrations

The status code registry has been updated with the registrations below:

Value	Description	Reference
100	Continue	Section 6.2.1
101	Switching Protocols	Section 6.2.2
200	OK	Section 6.3.1
201	Created	Section 6.3.2
202	Accepted	Section 6.3.3
203	Non-Authoritative Information	Section 6.3.4
204	No Content	Section 6.3.5
205	Reset Content	Section 6.3.6
300	Multiple Choices	Section 6.4.1
301	Moved Permanently	Section 6.4.2
302	Found	Section 6.4.3
303	See Other	Section 6.4.4
305	Use Proxy	Section 6.4.5
306	(Unused)	Section 6.4.6
307	Temporary Redirect	Section 6.4.7
400	Bad Request	Section 6.5.1
402	Payment Required	Section 6.5.2
403	Forbidden	Section 6.5.3
404	Not Found	Section 6.5.4
405	Method Not Allowed	Section 6.5.5
406	Not Acceptable	Section 6.5.6
408	Request Timeout	Section 6.5.7
409	Conflict	Section 6.5.8
410	Gone	Section 6.5.9
411	Length Required	Section 6.5.10
413	Payload Too Large	Section 6.5.11
414	URI Too Long	Section 6.5.12
415	Unsupported Media Type	Section 6.5.13
417	Expectation Failed	Section 6.5.14
426	Upgrade Required	Section 6.5.15
500	Internal Server Error	Section 6.6.1
501	Not Implemented	Section 6.6.2
502	Bad Gateway	Section 6.6.3
503	Service Unavailable	Section 6.6.4
504	Gateway Timeout	Section 6.6.5
505	HTTP Version Not Supported	Section 6.6.6

8.3. Header Field Registry

HTTP header fields are registered within the "Message Headers" registry located at <http://www.iana.org/assignments/message-headers>, as defined by [BCP90].

8.3.1. Considerations for New Header Fields

Header fields are key:value pairs that can be used to communicate data about the message, its payload, the target resource, or the connection (i.e., control data). See Section 3.2 of [RFC7230] for a general definition of header field syntax in HTTP messages.

The requirements for header field names are defined in [BCP90].

Authors of specifications defining new fields are advised to keep the name as short as practical and not to prefix the name with "X-" unless the header field will never be used on the Internet. (The "X-" prefix idiom has been extensively misused in practice; it was intended to only be used as a mechanism for avoiding name collisions inside proprietary software or intranet processing, since the prefix would ensure that private names never collide with a newly registered Internet name; see [BCP178] for further information).

New header field values typically have their syntax defined using ABNF ([RFC5234]), using the extension defined in Section 7 of [RFC7230] as necessary, and are usually constrained to the range of US-ASCII characters. Header fields needing a greater range of characters can use an encoding such as the one defined in [RFC5987].

Leading and trailing whitespace in raw field values is removed upon field parsing (Section 3.2.4 of [RFC7230]). Field definitions where leading or trailing whitespace in values is significant will have to use a container syntax such as quoted-string (Section 3.2.6 of [RFC7230]).

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. Typically, components that might contain a comma are protected with double-quotes using the quoted-string ABNF production.

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 3.1.1.5). 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 3.1.1.1).

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 [RFC7230]).

If it does not use the list syntax, document how to treat messages where the field occurs multiple times (a sensible default would be to ignore the 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 field's definition not allowing the list syntax. 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).

- o 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, etc.
- o Whether the field should be stored by origin servers that understand it upon a PUT request.
- o Whether the field semantics are further refined by the context, such as by existing request methods or status codes.
- o 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 [RFC7230]).
- o Under what conditions intermediaries are allowed to insert, delete, or modify the field's value.

- o Whether it is appropriate to list the field-name in a Vary response header field (e.g., when the request header field is used by an origin server's content selection algorithm; see Section 7.1.4).
- o Whether the header field is useful or allowable in trailers (see Section 4.1 of [RFC7230]).
- o Whether the header field ought to be preserved across redirects.
- o Whether it introduces any additional security considerations, such as disclosure of privacy-related data.

8.3.2. Registrations

The "Message Headers" registry has been updated with the following permanent registrations:

Header Field Name	Protocol	Status	Reference
Accept	http	standard	Section 5.3.2
Accept-Charset	http	standard	Section 5.3.3
Accept-Encoding	http	standard	Section 5.3.4
Accept-Language	http	standard	Section 5.3.5
Allow	http	standard	Section 7.4.1
Content-Encoding	http	standard	Section 3.1.2.2
Content-Language	http	standard	Section 3.1.3.2
Content-Location	http	standard	Section 3.1.4.2
Content-Type	http	standard	Section 3.1.1.5
Date	http	standard	Section 7.1.1.2
Expect	http	standard	Section 5.1.1
From	http	standard	Section 5.5.1
Location	http	standard	Section 7.1.2
Max-Forwards	http	standard	Section 5.1.2
MIME-Version	http	standard	Appendix A.1
Referer	http	standard	Section 5.5.2
Retry-After	http	standard	Section 7.1.3
Server	http	standard	Section 7.4.2
User-Agent	http	standard	Section 5.5.3
Vary	http	standard	Section 7.1.4

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

8.4. Content Coding Registry

The "HTTP Content Coding Registry" defines the namespace for content coding names (Section 4.2 of [RFC7230]). The content coding registry is maintained at <<http://www.iana.org/assignments/http-parameters>>.

8.4.1. Procedure

Content coding registrations MUST include the following fields:

- o Name
- o Description
- o Pointer to specification text

Names of content codings MUST NOT overlap with names of transfer codings (Section 4 of [RFC7230]), unless the encoding transformation is identical (as is the case for the compression codings defined in Section 4.2 of [RFC7230]).

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

8.4.2. Registrations

The "HTTP Content Coding Registry" has been updated with the registrations below:

Name	Description	Reference
identity	Reserved (synonym for "no encoding" in Accept-Encoding)	Section 5.3.4

9. Security Considerations

This section is meant to inform developers, information providers, and users of known security concerns relevant to HTTP semantics and its use for transferring information over the Internet. Considerations related to message syntax, parsing, and routing are discussed in Section 9 of [RFC7230].

The list of considerations below is not exhaustive. Most security concerns related to HTTP semantics are about securing server-side applications (code behind the HTTP interface), securing user agent

processing of payloads received via HTTP, or secure use of the Internet in general, rather than security of the protocol. Various organizations maintain topical information and links to current research on Web application security (e.g., [OWASP]).

9.1. Attacks Based on File and Path Names

Origin servers frequently make use of their local file system to manage the mapping from effective request URI to resource representations. Most file systems are not designed to protect against malicious file or path names. Therefore, an origin server needs to avoid accessing names that have a special significance to the system when mapping the request target to files, folders, or directories.

For example, UNIX, Microsoft Windows, and other operating systems use "." as a path component to indicate a directory level above the current one, and they use specially named paths or file names to send data to system devices. Similar naming conventions might exist within other types of storage systems. Likewise, local storage systems have an annoying tendency to prefer user-friendliness over security when handling invalid or unexpected characters, recomposition of decomposed characters, and case-normalization of case-insensitive names.

Attacks based on such special names tend to focus on either denial-of-service (e.g., telling the server to read from a COM port) or disclosure of configuration and source files that are not meant to be served.

9.2. Attacks Based on Command, Code, or Query Injection

Origin servers often use parameters within the URI as a means of identifying system services, selecting database entries, or choosing a data source. However, data received in a request cannot be trusted. An attacker could construct any of the request data elements (method, request-target, header fields, or body) to contain data that might be misinterpreted as a command, code, or query when passed through a command invocation, language interpreter, or database interface.

For example, SQL injection is a common attack wherein additional query language is inserted within some part of the request-target or header fields (e.g., Host, Referer, etc.). If the received data is used directly within a SELECT statement, the query language might be interpreted as a database command instead of a simple string value. This type of implementation vulnerability is extremely common, in spite of being easy to prevent.

In general, resource implementations ought to avoid use of request data in contexts that are processed or interpreted as instructions. Parameters ought to be compared to fixed strings and acted upon as a result of that comparison, rather than passed through an interface that is not prepared for untrusted data. Received data that isn't based on fixed parameters ought to be carefully filtered or encoded to avoid being misinterpreted.

Similar considerations apply to request data when it is stored and later processed, such as within log files, monitoring tools, or when included within a data format that allows embedded scripts.

9.3. Disclosure of Personal Information

Clients are often privy to large amounts of personal information, including both information provided by the user to interact with resources (e.g., the user's name, location, mail address, passwords, encryption keys, etc.) and information about the user's browsing activity over time (e.g., history, bookmarks, etc.). Implementations need to prevent unintentional disclosure of personal information.

9.4. Disclosure of Sensitive Information in URIs

URIs are intended to be shared, not secured, even when they identify secure resources. URIs are often shown on displays, added to templates when a page is printed, and stored in a variety of unprotected bookmark lists. It is therefore unwise to include information within a URI that is sensitive, personally identifiable, or a risk to disclose.

Authors of services ought to avoid 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 ought to use POST-based form submission instead.

Since the Referer header field tells a target site about the context that resulted in a request, it has the potential to reveal information about the user's immediate browsing history and any personal information that might be found in the referring resource's URI. Limitations on the Referer header field are described in Section 5.5.2 to address some of its security considerations.

9.5. Disclosure of Fragment after Redirects

Although fragment identifiers used within URI references are not sent in requests, implementers ought to be aware that they will be visible to the user agent and any extensions or scripts running as a result of the response. In particular, when a redirect occurs and the original request's fragment identifier is inherited by the new reference in Location (Section 7.1.2), this might have the effect of disclosing one site's fragment to another site. If the first site uses personal information in fragments, it ought to ensure that redirects to other sites include a (possibly empty) fragment component in order to block that inheritance.

9.6. Disclosure of Product Information

The User-Agent (Section 5.5.3), Via (Section 5.7.1 of [RFC7230]), and Server (Section 7.4.2) header fields often reveal information about the respective sender's software systems. In theory, this can make it easier for an attacker to exploit known security holes; in practice, attackers tend to try all potential holes regardless of the apparent software versions being used.

Proxies that serve as a portal through a network firewall ought to take special precautions regarding the transfer of header information that might identify hosts behind the firewall. The Via header field allows intermediaries to replace sensitive machine names with pseudonyms.

9.7. Browser Fingerprinting

Browser fingerprinting is a set of techniques for identifying a specific user agent over time through its unique set of characteristics. These characteristics might include information related to its TCP behavior, feature capabilities, and scripting environment, though of particular interest here is the set of unique characteristics that might be communicated via HTTP. Fingerprinting is considered a privacy concern because it enables tracking of a user agent's behavior over time without the corresponding controls that the user might have over other forms of data collection (e.g., cookies). Many general-purpose user agents (i.e., Web browsers) have taken steps to reduce their fingerprints.

There are a number of request header fields that might reveal information to servers that is sufficiently unique to enable fingerprinting. The From header field is the most obvious, though it is expected that From will only be sent when self-identification is desired by the user. Likewise, Cookie header fields are deliberately

designed to enable re-identification, so fingerprinting concerns only apply to situations where cookies are disabled or restricted by the user agent's configuration.

The User-Agent header field might contain enough information to uniquely identify a specific device, usually when combined with other characteristics, particularly if the user agent sends excessive details about the user's system or extensions. However, the source of unique information that is least expected by users is proactive negotiation (Section 5.3), including the Accept, Accept-Charset, Accept-Encoding, and Accept-Language header fields.

In addition to the fingerprinting concern, detailed use of the Accept-Language header field can reveal information the user might consider to be of a private nature. For example, understanding a given language set might be strongly correlated to membership in a particular ethnic group. An approach that limits such loss of privacy would be for a user agent to omit the sending of Accept-Language except for sites that have been whitelisted, perhaps via interaction after detecting a Vary header field that indicates language negotiation might be useful.

In environments where proxies are used to enhance privacy, user agents ought to be conservative in sending proactive negotiation header fields. General-purpose user agents that provide a high degree of header field configurability ought to inform users about the loss of privacy that might result if too much detail is provided. As an extreme privacy measure, proxies could filter the proactive negotiation header fields in relayed requests.

10. Acknowledgments

See Section 10 of [RFC7230].

11. References

11.1. Normative References

- [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.
- [RFC6365] Hoffman, P. and J. Klensin, "Terminology Used in Internationalization in the IETF", BCP 166, RFC 6365, September 2011.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.
- [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", RFC 7232, June 2014.
- [RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", RFC 7233, June 2014.
- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014.
- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, June 2014.

11.2. Informative References

- [BCP13] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, January 2013.
- [BCP178] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, June 2012.

- [BCP90] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.
- [OWASP] van der Stock, A., Ed., "A Guide to Building Secure Web Applications and Web Services", The Open Web Application Security Project (OWASP) 2.0.1, July 2005, <<https://www.owasp.org/>>.
- [REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Doctoral Dissertation, University of California, Irvine, September 2000, <<http://roy.gbiv.com/pubs/dissertation/top.htm>>.
- [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.
- [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.
- [RFC2774] Frystyk, H., Leach, P., and S. Lawrence, "An HTTP Extension Framework", RFC 2774, February 2000.
- [RFC2817] Khare, R. and S. Lawrence, "Upgrading to TLS Within HTTP/1.1", RFC 2817, May 2000.
- [RFC2978] Freed, N. and J. Postel, "IANA Charset Registration Procedures", BCP 19, RFC 2978, October 2000.

- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 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.
- [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.
- [RFC5987] Reschke, J., "Character Set and Language Encoding for Hypertext Transfer Protocol (HTTP) Header Field Parameters", RFC 5987, August 2010.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.
- [RFC6265] Barth, A., "HTTP State Management Mechanism", RFC 6265, April 2011.
- [RFC6266] Reschke, J., "Use of the Content-Disposition Header Field in the Hypertext Transfer Protocol (HTTP)", RFC 6266, June 2011.
- [RFC7238] Reschke, J., "The Hypertext Transfer Protocol (HTTP) Status Code 308 (Permanent Redirect)", RFC 7238, June 2014.

Appendix A. Differences between HTTP and MIME

HTTP/1.1 uses many of the constructs defined for the Internet Message Format [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 header fields. However, RFC 2045 is focused only on email; applications of HTTP have many characteristics that differ from email; hence, HTTP has features that differ from 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 and from strict MIME environments need to be aware of these differences and provide the appropriate conversions where necessary.

A.1. MIME-Version

HTTP is not a MIME-compliant protocol. However, messages can 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]). Senders are responsible for ensuring full conformance (where possible) when exporting HTTP messages to strict MIME environments.

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 3.1.1.3 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.

A proxy or gateway from HTTP to a strict MIME environment ought to translate all line breaks within the text media types described in Section 3.1.1.3 of this document to the RFC 2049 canonical form of CRLF. Note, however, this might be complicated by the presence of a Content-Encoding and by the fact that HTTP allows the use of some charsets that do not use octets 13 and 10 to represent CR and LF, respectively.

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 7.1.1.1) to simplify the process of date comparison. Proxies and gateways from other protocols ought to 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. Conversion 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 ought to 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. Conversion of Content-Transfer-Encoding

HTTP does not use the Content-Transfer-Encoding field of MIME. Proxies and gateways from MIME-compliant protocols to HTTP need to 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 ought to transform and 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 that 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 transfers message-bodies as

payload and, aside from the "multipart/byteranges" type (Appendix A of [RFC7233]), does not interpret the content or any MIME header lines that might be contained therein.

Appendix B. Changes from RFC 2616

The primary changes in this revision have been editorial in nature: extracting the messaging syntax and partitioning HTTP semantics into separate documents for the core features, conditional requests, partial requests, caching, and authentication. The conformance language has been revised to clearly target requirements and the terminology has been improved to distinguish payload from representations and representations from resources.

A new requirement has been added that semantics embedded in a URI be disabled when those semantics are inconsistent with the request method, since this is a common cause of interoperability failure. (Section 2)

An algorithm has been added for determining if a payload is associated with a specific identifier. (Section 3.1.4.1)

The default charset of ISO-8859-1 for text media types has been removed; the default is now whatever the media type definition says. Likewise, special treatment of ISO-8859-1 has been removed from the Accept-Charset header field. (Section 3.1.1.3 and Section 5.3.3)

The definition of Content-Location has been changed to no longer affect the base URI for resolving relative URI references, due to poor implementation support and the undesirable effect of potentially breaking relative links in content-negotiated resources. (Section 3.1.4.2)

To be consistent with the method-neutral parsing algorithm of [RFC7230], the definition of GET has been relaxed so that requests can have a body, even though a body has no meaning for GET. (Section 4.3.1)

Servers are no longer required to handle all Content-* header fields and use of Content-Range has been explicitly banned in PUT requests. (Section 4.3.4)

Definition of the CONNECT method has been moved from [RFC2817] to this specification. (Section 4.3.6)

The OPTIONS and TRACE request methods have been defined as being safe. (Section 4.3.7 and Section 4.3.8)

The Expect header field's extension mechanism has been removed due to widely-deployed broken implementations. (Section 5.1.1)

The Max-Forwards header field has been restricted to the OPTIONS and TRACE methods; previously, extension methods could have used it as well. (Section 5.1.2)

The "about:blank" URI has been suggested as a value for the Referer header field when no referring URI is applicable, which distinguishes that case from others where the Referer field is not sent or has been removed. (Section 5.5.2)

The following status codes are now cacheable (that is, they can be stored and reused by a cache without explicit freshness information present): 204, 404, 405, 414, 501. (Section 6)

The 201 (Created) status description has been changed to allow for the possibility that more than one resource has been created. (Section 6.3.2)

The definition of 203 (Non-Authoritative Information) has been broadened to include cases of payload transformations as well. (Section 6.3.4)

The set of request methods that are safe to automatically redirect is no longer closed; user agents are able to make that determination based upon the request method semantics. The redirect status codes 301, 302, and 307 no longer have normative requirements on response payloads and user interaction. (Section 6.4)

The status codes 301 and 302 have been changed to allow user agents to rewrite the method from POST to GET. (Sections 6.4.2 and 6.4.3)

The description of the 303 (See Other) status code has been changed to allow it to be cached if explicit freshness information is given, and a specific definition has been added for a 303 response to GET. (Section 6.4.4)

The 305 (Use Proxy) status code has been deprecated due to security concerns regarding in-band configuration of a proxy. (Section 6.4.5)

The 400 (Bad Request) status code has been relaxed so that it isn't limited to syntax errors. (Section 6.5.1)

The 426 (Upgrade Required) status code has been incorporated from [RFC2817]. (Section 6.5.15)

The target of requirements on HTTP-date and the Date header field have been reduced to those systems generating the date, rather than all systems sending a date. (Section 7.1.1)

The syntax of the Location header field has been changed to allow all URI references, including relative references and fragments, along with some clarifications as to when use of fragments would not be appropriate. (Section 7.1.2)

Allow has been reclassified as a response header field, removing the option to specify it in a PUT request. Requirements relating to the content of Allow have been relaxed; correspondingly, clients are not required to always trust its value. (Section 7.4.1)

A Method Registry has been defined. (Section 8.1)

The Status Code Registry has been redefined by this specification; previously, it was defined in Section 7.1 of [RFC2817]. (Section 8.2)

Registration of content codings has been changed to require IETF Review. (Section 8.4)

The Content-Disposition header field has been removed since it is now defined by [RFC6266].

The Content-MD5 header field has been removed because it was inconsistently implemented with respect to partial responses.

Appendix C. 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 [RFC7230]:

BWS	= <BWS, see [RFC7230], Section 3.2.3>
OWS	= <OWS, see [RFC7230], Section 3.2.3>
RWS	= <RWS, see [RFC7230], Section 3.2.3>
URI-reference	= <URI-reference, see [RFC7230], Section 2.7>
absolute-URI	= <absolute-URI, see [RFC7230], Section 2.7>
comment	= <comment, see [RFC7230], Section 3.2.6>
field-name	= <comment, see [RFC7230], Section 3.2>
partial-URI	= <partial-URI, see [RFC7230], Section 2.7>

quoted-string = <quoted-string, see [RFC7230], Section 3.2.6>
 token = <token, see [RFC7230], Section 3.2.6>

Appendix D. Collected ABNF

In the collected ABNF below, list rules are expanded as per Section 1.2 of [RFC7230].

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

BWS = <BWS, see [RFC7230], Section 3.2.3>

```
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 = "100-continue"

From = mailbox

GMT = %x47.4D.54 ; GMT

HTTP-date = IMF-fixdate / obs-date

IMF-fixdate = day-name "," SP date1 SP time-of-day SP GMT

Location = URI-reference

Max-Forwards = 1*DIGIT

OWS = <OWS, see [RFC7230], Section 3.2.3>

RWS = <RWS, see [RFC7230], Section 3.2.3>

Referer = absolute-URI / partial-URI

Retry-After = HTTP-date / delay-seconds

```

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

URI-reference = <URI-reference, see [RFC7230], Section 2.7>
User-Agent = product *( RWS ( product / comment ) )

Vary = "*" / ( *( "," OWS ) field-name *( OWS "," [ OWS field-name ]
) )

absolute-URI = <absolute-URI, see [RFC7230], Section 2.7>
accept-ext = OWS ";" OWS token [ "=" ( token / quoted-string ) ]
accept-params = weight *accept-ext
asctime-date = day-name SP date3 SP time-of-day SP year

charset = token
codings = content-coding / "identity" / "*"
comment = <comment, see [RFC7230], Section 3.2.6>
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-1 = %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
delay-seconds = 1*DIGIT

field-name = <comment, see [RFC7230], Section 3.2>

hour = 2DIGIT

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

mailbox = <mailbox, see [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 = token "=" ( token / quoted-string )
partial-URI = <partial-URI, see [RFC7230], Section 2.7>
product = token [ "/" product-version ]
product-version = token
quoted-string = <quoted-string, see [RFC7230], Section 3.2.6>
qvalue = ( "0" [ "." *3DIGIT ] ) / ( "1" [ "." *3"0" ] )

rfc850-date = day-name-1 "," SP date2 SP time-of-day SP GMT

second = 2DIGIT
subtype = token

time-of-day = hour ":" minute ":" second
token = <token, see [RFC7230], Section 3.2.6>
type = token

weight = OWS ";" OWS "q=" qvalue

year = 4DIGIT
```

Index

- 1
 - 1xx Informational (status code class) 50
- 2
 - 2xx Successful (status code class) 51
- 3
 - 3xx Redirection (status code class) 54
- 4
 - 4xx Client Error (status code class) 58
- 5
 - 5xx Server Error (status code class) 62
- 1
 - 100 Continue (status code) 50
 - 100-continue (expect value) 34
 - 101 Switching Protocols (status code) 50
- 2
 - 200 OK (status code) 51
 - 201 Created (status code) 52
 - 202 Accepted (status code) 52
 - 203 Non-Authoritative Information (status code) 52
 - 204 No Content (status code) 53
 - 205 Reset Content (status code) 53
- 3
 - 300 Multiple Choices (status code) 55
 - 301 Moved Permanently (status code) 56
 - 302 Found (status code) 56
 - 303 See Other (status code) 57
 - 305 Use Proxy (status code) 58
 - 306 (Unused) (status code) 58
 - 307 Temporary Redirect (status code) 58
- 4
 - 400 Bad Request (status code) 58
 - 402 Payment Required (status code) 59
 - 403 Forbidden (status code) 59
 - 404 Not Found (status code) 59
 - 405 Method Not Allowed (status code) 59
 - 406 Not Acceptable (status code) 59
 - 408 Request Timeout (status code) 60
 - 409 Conflict (status code) 60

410 Gone (status code)	60
411 Length Required (status code)	61
413 Payload Too Large (status code)	61
414 URI Too Long (status code)	61
415 Unsupported Media Type (status code)	62
417 Expectation Failed (status code)	62
426 Upgrade Required (status code)	62
5	
500 Internal Server Error (status code)	63
501 Not Implemented (status code)	63
502 Bad Gateway (status code)	63
503 Service Unavailable (status code)	63
504 Gateway Timeout (status code)	63
505 HTTP Version Not Supported (status code)	64
A	
Accept header field	38
Accept-Charset header field	40
Accept-Encoding header field	41
Accept-Language header field	42
Allow header field	72
C	
cacheable	24
compress (content coding)	11
conditional request	36
CONNECT method	30
content coding	11
content negotiation	6
Content-Encoding header field	12
Content-Language header field	13
Content-Location header field	15
Content-Transfer-Encoding header field	89
Content-Type header field	10
D	
Date header field	67
deflate (content coding)	11
DELETE method	29
E	
Expect header field	34
F	
From header field	44

G

GET method 24
Grammar
 Accept 38
 Accept-Charset 40
 Accept-Encoding 41
 accept-ext 38
 Accept-Language 42
 accept-params 38
 Allow 72
 asctime-date 66
 charset 9
 codings 41
 content-coding 11
 Content-Encoding 12
 Content-Language 13
 Content-Location 15
 Content-Type 10
 Date 67
 date1 65
 day 65
 day-name 65
 day-name-1 65
 delay-seconds 69
 Expect 34
 From 44
 GMT 65
 hour 65
 HTTP-date 65
 IMF-fixdate 65
 language-range 42
 language-tag 13
 Location 68
 Max-Forwards 36
 media-range 38
 media-type 8
 method 21
 minute 65
 month 65
 obs-date 66
 parameter 8
 product 46
 product-version 46
 qvalue 38
 Referer 45
 Retry-After 69
 rfc850-date 66
 second 65

Server 73
subtype 8
time-of-day 65
type 8
User-Agent 46
Vary 70
weight 38
year 65
gzip (content coding) 11

H
HEAD method 25

I
idempotent 23

L
Location header field 68

M
Max-Forwards header field 36
MIME-Version header field 89

O
OPTIONS method 31

P
payload 17
POST method 25
PUT method 26

R
Referer header field 45
representation 7
Retry-After header field 69

S
safe 22
selected representation 7, 71
Server header field 73
Status Codes Classes
1xx Informational 50
2xx Successful 51
3xx Redirection 54
4xx Client Error 58
5xx Server Error 62

T	
TRACE method	32
U	
User-Agent header field	46
V	
Vary header field	70
X	
x-compress (content coding)	11
x-gzip (content coding)	11

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/>

Julian F. Reschke (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
Germany

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