HTTPbis Working Group
Internet-Draft
Obsoletes: 2616 (if approved)
Intended status: Standards Track
Expires: April 28, 2011

R. Fielding, Ed.
Day Software
J. Gettys
Alcatel-Lucent
J. Mogul
HP
H. Frystyk
Microsoft
L. Masinter
Adobe Systems
P. Leach
Microsoft
T. Berners-Lee
W3C/MIT
Y. Lafon, Ed.
W3C
J. Reschke, Ed.
greenbytes

October 25, 2010

# HTTP/1.1, part 3: Message Payload and Content Negotiation
## draft-ietf-httpbis-p3-payload-12

## 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 3 of the seven-part specification that defines the protocol referred to as "HTTP/1.1" and, taken together, obsoletes RFC 2616. Part 3 defines HTTP message content, metadata, and content negotiation.

## 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://tools.ietf.org/wg/httpbis/trac/report/3> and related documents (including fancy diffs) can be found at <http://tools.ietf.org/wg/httpbis/>.

The changes in this draft are summarized in Appendix E.13.

## Status of This Memo

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

## Copyright Notice



## Table of Contents

# 1. Introduction

This document defines HTTP/1.1 message payloads (a.k.a., content), the associated metadata header fields that define how the payload is intended to be interpreted by a recipient, the request header fields that might influence content selection, and the various selection algorithms that are collectively referred to as HTTP content negotiation.

This document is currently disorganized in order to minimize the changes between drafts and enable reviewers to see the smaller errata changes. A future draft will reorganize the sections to better reflect the content. In particular, the sections on entities will be renamed payload and moved to the first half of the document, while the sections on content negotiation and associated request header fields will be moved to the second half. The current mess reflects how widely dispersed these topics and associated requirements had become in [RFC2616].

## 1.1. Terminology

This specification uses a number of terms to refer to the roles

played by participants in, and objects of, the HTTP communication.

content negotiation

The mechanism for selecting the appropriate representation when servicing a request. The representation in any response can be negotiated (including error responses).

## 1.2. Requirements

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

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





## 1.3. Syntax Notation

This specification uses the ABNF syntax defined in Section 1.2 of [Part1] (which extends the syntax defined in [RFC5234] with a list rule). Appendix D shows the collected ABNF, with the list rule expanded.

The following core rules are included by reference, as defined in [RFC5234], Appendix B.1: ALPHA (letters), CR (carriage return), CRLF (CR LF), CTL (controls), DIGIT (decimal 0-9), DQUOTE (double quote), HEXDIG (hexadecimal 0-9/A-F/a-f), LF (line feed), OCTET (any 8-bit sequence of data), SP (space), VCHAR (any visible USASCII character), and WSP (whitespace).

### 1.3.1. Core Rules

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

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

### 1.3.2. ABNF Rules defined in other Parts of the Specification

The ABNF rules below are defined in other parts:

```
  absolute-URI   = <absolute-URI, defined in [Part1], Section 2.6>
  Content-Length = <Content-Length, defined in [Part1], Section 9.2>
  partial-URI    = <partial-URI, defined in [Part1], Section 2.6>
  qvalue         = <qvalue, defined in [Part1], Section 6.4>


  Last-Modified  = <Last-Modified, defined in [Part4], Section 6.6>


  Content-Range  = <Content-Range, defined in [Part5], Section 5.2>


  Expires        = <Expires, defined in [Part6], Section 3.3>
```

## 2. Protocol Parameters

### 2.1. Character Sets

HTTP uses the same definition of the term "character set" as that described for MIME:

The term "character set" is used in this document to refer to a





method used with one or more tables to convert a sequence of octets into a sequence of characters. Note that unconditional conversion in the other direction is not required, in that not all characters might be available in a given character set and a character set might provide more than one sequence of octets to represent a particular character. This definition is intended to allow various kinds of character encoding, from simple single-table mappings such as US-ASCII to complex table switching methods such as those that use ISO-

2022's techniques. However, the definition associated with a MIME character set name MUST fully specify the mapping to be performed from octets to characters. In particular, use of external profiling information to determine the exact mapping is not permitted.

> Note: This use of the term "character set" is more commonly referred to as a "character encoding". However, since HTTP and MIME share the same registry, it is important that the terminology also be shared.

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

```
  charset = token
```

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

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

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

### 2.1.1. Missing Charset

Some HTTP/1.0 software has interpreted a Content-Type header field without charset parameter incorrectly to mean "recipient should guess". Senders wishing to defeat this behavior MAY include a charset parameter even when the charset is ISO-8859-1 ([ISO-8859-1]) and SHOULD do so when it is known that it will not confuse the recipient.





Unfortunately, some older HTTP/1.0 clients did not deal properly with

an explicit charset parameter. HTTP/1.1 recipients MUST respect the charset label provided by the sender; and those user agents that have a provision to "guess" a charset MUST use the charset from the content-type field if they support that charset, rather than the recipient's preference, when initially displaying a document. See Section 2.3.1.

## 2.2. Content Codings

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

```
  content-coding   = token
```

All content-coding values are case-insensitive. HTTP/1.1 uses content-coding values in the Accept-Encoding (Section 6.3) and Content-Encoding (Section 6.5) header fields. Although the value describes the content-coding, what is more important is that it indicates what decoding mechanism will be required to remove the encoding.

compress

See Section 6.2.2.1 of [Part1].

deflate

See Section 6.2.2.2 of [Part1].

gzip

See Section 6.2.2.3 of [Part1].

identity

The default (identity) encoding; the use of no transformation whatsoever. This content-coding is used only in the Accept-Encoding header field, and SHOULD NOT be used in the Content-Encoding header field.

### 2.2.1. Content Coding Registry

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

Registrations MUST include the following fields:

- Name
- Description
- Pointer to specification text

Names of content codings MUST NOT overlap with names of transfer codings (Section 6.2 of [Part1]), unless the encoding transformation is identical (as it is the case for the compression codings defined in Section 6.2.2 of [Part1]).

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

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

## 2.3. Media Types

HTTP uses Internet Media Types [RFC2046] in the Content-Type (Section 6.9) and Accept (Section 6.1) header fields in order to provide open and extensible data typing and type negotiation.

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

Parameters MAY follow the type/subtype in the form of attribute/value pairs.

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

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

media-type, depending on its definition within the media type registry.





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

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

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

### 2.3.1. Canonicalization and Text Defaults

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

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

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

The "charset" parameter is used with some media types to define the

character encoding (Section 2.1) of the data. When no explicit charset parameter is provided by the sender, media subtypes of the "text" type are defined to have a default charset value of "ISO-8859-1" when received via HTTP. Data in character encodings other than "ISO-8859-1" or its subsets MUST be labeled with an appropriate charset value. See Section 2.1.1 for compatibility problems.





### 2.3.2. Multipart Types

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

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

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

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

## 2.4. Language Tags

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

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

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

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

Example tags include:

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





See [RFC5646] for further information.

## 3. Payload

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

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

### 3.1. Payload Header Fields

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

```
   Content-Length           ; [Part1], Section 9.2
   Content-MD5              ; Section 6.8
```

```
    Content-Range             ; [Part5], Section 5.2
```

## 3.2. Payload Body

A payload body is only present in a message when a message-body is present, as described in Section 3.3 of [Part1]. The payload body is obtained from the message-body by decoding any Transfer-Encoding that might have been applied to ensure safe and proper transfer of the message.

# 4. Representation

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

Most, but not all, representations transferred via HTTP are intended to be a representation of the target resource (the resource identified by the effective request URI). The precise semantics of a representation are determined by the type of message (request or response), the request method, the response status code, and the





representation metadata. For example, the above semantic is true for the representation in any 200 (OK) response to GET and for the representation in any PUT request. A 200 response to PUT, in contrast, contains either a representation that describes the successful action or a representation of the target resource, with the latter indicated by a Content-Location header field with the same value as the effective request URI. Likewise, response messages with an error status code usually contain a representation that describes the error and what next steps are suggested for resolving it.

## 4.1. Representation Header Fields

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

The following header fields are defined as representation metadata:

```
   Content-Encoding          ; Section 6.5
   Content-Language          ; Section 6.6
   Content-Location          ; Section 6.7
   Content-Type              ; Section 6.9
   Expires                   ; [Part6], Section 3.3
   Last-Modified             ; [Part4], Section 6.6
```

## 4.2. Representation Data

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

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

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

Content-Type specifies the media type of the underlying data, which defines both the data format and how that data SHOULD be processed by the recipient (within the scope of the request method semantics). Any HTTP/1.1 message containing a payload body SHOULD include a Content-Type header field defining the media type of the associated representation unless that metadata is unknown to the sender. If the Content-Type header field is not present, it indicates that the





sender does not know the media type of the representation; recipients MAY either assume that the media type is "application/octet-stream" ([RFC2046], Section 4.5.1) or examine the content to determine its type.

In practice, resource owners do not always properly configure their origin server to provide the correct Content-Type for a given representation, with the result that some clients will examine a response body's content and override the specified type. Clients that do so risk drawing incorrect conclusions, which might expose additional security risks (e.g., "privilege escalation").

Furthermore, it is impossible to determine the sender's intent by examining the data format: many data formats match multiple media types that differ only in processing semantics. Implementers are encouraged to provide a means of disabling such "content sniffing" when it is used.

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

## 5. Content Negotiation

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

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

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





proposed.

These patterns are all widely used, and have trade-offs in applicability and practicality. In particular, when the number of

preferences or capabilities to be expressed by a client are large (such as when many different formats are supported by a user-agent), server-driven negotiation becomes unwieldy, and might not be appropriate. Conversely, when the number of representations to choose from is very large, agent-driven negotiation might not be appropriate.

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

## 5.1. Server-driven Negotiation

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

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

Server-driven negotiation has disadvantages:

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

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

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

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

HTTP/1.1 includes the following request-header fields for enabling server-driven negotiation through description of user agent capabilities and user preferences: Accept (Section 6.1), Accept-Charset (Section 6.2), Accept-Encoding (Section 6.3), Accept-Language (Section 6.4), and User-Agent (Section 9.9 of [Part2]). However, an origin server is not limited to these dimensions and MAY vary the response based on any aspect of the request, including information outside the request-header fields or within extension header fields not defined by this specification.

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

The Vary header field (Section 3.5 of [Part6]) can be used to express the parameters the server uses to select a representation that is subject to server-driven negotiation.

## 5.2. Agent-driven Negotiation

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

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

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

HTTP/1.1.





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

## 6. Header Field Definitions

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

### 6.1. Accept

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

```
  Accept   = "Accept" ":" OWS Accept-v
  Accept-v = #( media-range [ accept-params ] )

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

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

Each media-range MAY be followed by one or more accept-params, beginning with the "q" parameter for indicating a relative quality factor. The first "q" parameter (if any) separates the media-range parameter(s) from the accept-params. Quality factors allow the user or user agent to indicate the relative degree of preference for that media-range, using the qvalue scale from 0 to 1 (Section 6.4 of [Part1]). The default value is q=1.

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





registering any parameter named "q".

The example

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

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

If no Accept header field is present, then it is assumed that the client accepts all media types. If an Accept header field is present, and if the server cannot send a response which is acceptable according to the combined Accept field value, then the server SHOULD send a 406 (Not Acceptable) response.

A more elaborate example is

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

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

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

```
Accept: text/*, text/html, text/html;level=1, */*
```

have the following precedence:

1. text/html;level=1

2. text/html

3. text/*

4. */*

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

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





would cause the following values to be associated:

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

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

## 6.2. Accept-Charset

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

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

Character set values are described in Section 2.1. Each charset MAY be given an associated quality value which represents the user's preference for that charset. The default value is q=1. An example is

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

The special value "*", if present in the Accept-Charset field, matches every character set (including ISO-8859-1) which is not mentioned elsewhere in the Accept-Charset field. If no "*" is present in an Accept-Charset field, then all character sets not explicitly mentioned get a quality value of 0, except for ISO-8859-1, which gets a quality value of 1 if not explicitly mentioned.

If no Accept-Charset header field is present, the default is that any character set is acceptable. If an Accept-Charset header field is present, and if the server cannot send a response which is acceptable





according to the Accept-Charset header field, then the server SHOULD send an error response with the 406 (Not Acceptable) status code, though the sending of an unacceptable response is also allowed.

## 6.3. Accept-Encoding

The "Accept-Encoding" request-header field can be used by user agents to indicate what response content-codings (Section 2.2) are acceptable in the response.

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

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

Examples of its use are:

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

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

1. If the content-coding is one of the content-codings listed in the Accept-Encoding field, then it is acceptable, unless it is accompanied by a qvalue of 0. (As defined in Section 6.4 of [Part1], a qvalue of 0 means "not acceptable".)

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

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

4. The "identity" content-coding is always acceptable, unless specifically refused because the Accept-Encoding field includes "identity;q=0", or because the field includes "*;q=0" and does not explicitly include the "identity" content-coding. If the Accept-Encoding field-value is empty, then only the "identity"





encoding is acceptable.

If an Accept-Encoding field is present in a request, and if the server cannot send a response which is acceptable according to the Accept-Encoding header field, then the server SHOULD send an error response with the 406 (Not Acceptable) status code.

If no Accept-Encoding field is present in a request, the server MAY assume that the client will accept any content coding. In this case, if "identity" is one of the available content-codings, then the server SHOULD use the "identity" content-coding, unless it has additional information that a different content-coding is meaningful

to the client.

Note: If the request does not include an Accept-Encoding field, and if the "identity" content-coding is unavailable, then content-codings commonly understood by HTTP/1.0 clients (i.e., "gzip" and "compress") are preferred; some older clients improperly display messages sent with other content-codings. The server might also make this decision based on information about the particular user-agent or client.

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

## 6.4. Accept-Language

The "Accept-Language" request-header field can be used by user agents to indicate the set of natural languages that are preferred in the response. Language tags are defined in Section 2.4.

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

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

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

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





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

Note: The "Basic Filtering" scheme ([RFC4647], Section 3.3.1) is

identical to the matching scheme that was previously defined in Section 14.4 of [RFC2616].

It might be contrary to the privacy expectations of the user to send an Accept-Language header field with the complete linguistic preferences of the user in every request. For a discussion of this issue, see Section 8.1.

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

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

## 6.5. Content-Encoding

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

```
Content-Encoding   = "Content-Encoding" ":" OWS Content-Encoding-v
Content-Encoding-v = 1#content-coding
```

Content codings are defined in Section 2.2. An example of its use is

```
Content-Encoding: gzip
```

The content-coding is a characteristic of the representation. Typically, the representation body is stored with this encoding and is only decoded before rendering or analogous usage. However, a non-transparent proxy MAY modify the content-coding if the new coding is

known to be acceptable to the recipient, unless the "no-transform" cache-control directive is present in the message.

If the content-coding of a representation is not "identity", then the representation metadata MUST include a Content-Encoding header field (Section 6.5) that lists the non-identity content-coding(s) used.

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

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

### 6.6. Content-Language

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

```
Content-Language   = "Content-Language" ":" OWS Content-Language-v
Content-Language-v = 1#language-tag
```

Language tags are defined in Section 2.4. The primary purpose of Content-Language is to allow a user to identify and differentiate representations according to the user's own preferred language. Thus, if the body content is intended only for a Danish-literate audience, the appropriate field is

```
Content-Language: da
```

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

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

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

However, just because multiple languages are present within a representation does not mean that it is intended for multiple

linguistic audiences. An example would be a beginner's language primer, such as "A First Lesson in Latin", which is clearly intended to be used by an English-literate audience. In this case, the Content-Language would properly only include "en".

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

## 6.7. Content-Location

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

```
  Content-Location   = "Content-Location" ":" OWS
                       Content-Location-v
  Content-Location-v =
                       absolute-URI / partial-URI
```

The Content-Location value is not a replacement for the effective Request URI (Section 4.3 of [Part1]). It is representation metadata. It has the same syntax and semantics as the header field of the same name defined for MIME body parts in Section 4 of [RFC2557]. However, its appearance in an HTTP message has some special implications for HTTP recipients.

If Content-Location is included in a response message and its value is the same as the effective request URI, then the response payload SHOULD be considered the current representation of that resource. For a GET or HEAD request, this is the same as the default semantics when no Content-Location is provided by the server. For a state-changing method like PUT or POST, it implies that the server's response contains the new representation of that resource, thereby distinguishing it from representations that might only report about the action (e.g., "It worked!"). This allows authoring applications to update their local copies without the need for a subsequent GET request.

If Content-Location is included in a response message and its value differs from the effective request URI, then the origin server is

informing recipients that this representation has its own, presumably more specific, identifier. For a GET or HEAD request, this is an indication that the effective request URI identifies a resource that is subject to content negotiation and the representation selected for this response can also be found at the identified URI. For other methods, such a Content-Location indicates that this representation





contains a report on the action's status and the same report is available (for future access with GET) at the given URI. For example, a purchase transaction made via the POST method might include a receipt document as the payload of the 200 response; the Content-Location value provides an identifier for retrieving a copy of that same receipt in the future.

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

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

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

If the Content-Location value is a partial URI, the partial URI is

interpreted relative to the effective request URI.

## 6.8. Content-MD5

The "Content-MD5" header field, as defined in [RFC1864], is an MD5 digest of the payload body that provides an end-to-end message integrity check (MIC) of the payload body (the message-body after any transfer-coding is decoded). Note that a MIC is good for detecting accidental modification of the payload body in transit, but is not proof against malicious attacks.

```
  Content-MD5   = "Content-MD5" ":" OWS Content-MD5-v
  Content-MD5-v = <base64 of 128 bit MD5 digest as per [RFC1864]>
```





The Content-MD5 header field MAY be generated by an origin server or client to function as an integrity check of the payload body. Only origin servers or user agents MAY generate the Content-MD5 header field; proxies and gateways MUST NOT generate it, as this would defeat its value as an end-to-end integrity check. Any recipient MAY check that the digest value in this header field matches a corresponding digest calculated on payload body as received.

The MD5 digest is computed based on the content of the payload body, including any content-coding, but not including any transfer-coding applied to the message-body because such transfer-codings might be applied or removed anywhere along the request/response chain. If the message is received with a transfer-coding, that encoding MUST be decoded prior to checking the Content-MD5 value against the received payload.

HTTP extends RFC 1864 to permit the digest to be computed for MIME composite media-types (e.g., multipart/* and message/rfc822), but this does not change how the digest is computed as defined in the preceding paragraph.

There are several consequences of this. The payload for composite types MAY contain many body-parts, each with its own MIME and HTTP header fields (including Content-MD5, Content-Transfer-Encoding, and Content-Encoding header fields). If a body-part has a Content-Transfer-Encoding or Content-Encoding header field, it is assumed that the content of the body-part has had the encoding applied, and the body-part is included in the Content-MD5 digest as is -- i.e.,

after the application. The Transfer-Encoding header field is not allowed within body-parts.

Conversion of all line breaks to CRLF MUST NOT be done before computing or checking the digest: the line break convention used in the text actually transmitted MUST be left unaltered when computing the digest.

> Note: While the definition of Content-MD5 is exactly the same for HTTP as in RFC 1864 for MIME entity-bodies, there are several ways in which the application of Content-MD5 to HTTP entity-bodies differs from its application to MIME entity-bodies. One is that HTTP, unlike MIME, does not use Content-Transfer-Encoding, and does use Transfer-Encoding and Content-Encoding. Another is that HTTP more frequently uses binary content types than MIME, so it is worth noting that, in such cases, the byte order used to compute the digest is the transmission byte order defined for the type. Lastly, HTTP allows transmission of text types with any of several line break conventions and not just the canonical form using CRLF.





### 6.9. Content-Type

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

```
Content-Type   = "Content-Type" ":" OWS Content-Type-v
Content-Type-v = media-type
```

Media types are defined in Section 2.3. An example of the field is

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

Further discussion of Content-Type is provided in Section 4.2.

## 7. IANA Considerations

### 7.1. Header Field Registration

The Message Header Field Registry located at <http://www.iana.org/

assignments/message-headers/message-header-index.html> shall be updated with the permanent registrations below (see [RFC3864]):

| Header Field Name | Protocol | Status | Reference |
|---|---|---|---|
| Accept | http | standard | Section 6.1 |
| Accept-Charset | http | standard | Section 6.2 |
| Accept-Encoding | http | standard | Section 6.3 |
| Accept-Language | http | standard | Section 6.4 |
| Content-Encoding | http | standard | Section 6.5 |
| Content-Language | http | standard | Section 6.6 |
| Content-Location | http | standard | Section 6.7 |
| Content-MD5 | http | standard | Section 6.8 |
| Content-Type | http | standard | Section 6.9 |
| MIME-Version | http | standard | Appendix A.1 |

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

## 7.2. Content Coding Registry

The registration procedure for HTTP Content Codings is now defined by Section 2.2.1 of this document.

The HTTP Content Codings Registry located at





<http://www.iana.org/assignments/http-parameters> shall be updated with the registration below:

| Name | Description | Reference |
|---|---|---|
| compress | UNIX "compress" program method | Section 6.2.1 of [Part1] |
| deflate | "deflate" compression mechanism ([RFC1951]) used inside the "zlib" data format ([RFC1950]) | Section 6.2.2 of [Part1] |
| gzip | Same as GNU zip [RFC1952] | Section 6.2.3 of |

| | | |
|---|---|---|
| identity | No transformation | [Part1] Section 2.2 |

## 8. Security Considerations

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

### 8.1. Privacy Issues Connected to Accept Header Fields

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

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

Elaborate user-customized accept header fields sent in every request, in particular if these include quality values, can be used by servers as relatively reliable and long-lived user identifiers. Such user





identifiers would allow content providers to do click-trail tracking, and would allow collaborating content providers to match cross-server click-trails or form submissions of individual users. Note that for many users not behind a proxy, the network address of the host running the user agent will also serve as a long-lived user identifier. In environments where proxies are used to enhance privacy, user agents ought to be conservative in offering accept

header configuration options to end users. As an extreme privacy measure, proxies could filter the accept header fields in relayed requests. General purpose user agents which provide a high degree of header configurability SHOULD warn users about the loss of privacy which can be involved.

## 9. Acknowledgments

## 10. References

### 10.1. Normative References


[ISO-8859-1] International Organization for Standardization, "Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1", ISO/IEC 8859-1:1998, 1998.

[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-12 (work in progress), October 2010.

[Part2] 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 2: Message Semantics", draft-ietf-httpbis-p2-semantics-12 (work in progress), October 2010.

[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-12 (work in progress), October 2010.

[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-12 (work in progress), October 2010.

[Part6] Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching", draft-ietf-httpbis-p6-cache-12 (work in progress), October 2010.

[RFC1864] Myers, J. and M. Rose, "The Content-MD5 Header Field", RFC 1864, October 1995.

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

RFC 1950 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].

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

RFC 1951 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].

[RFC1952] Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-Pehrson, "GZIP file format specification version 4.3", RFC 1952, May 1996.

RFC 1952 is an Informational RFC, thus it might be less stable than this specification. On the other hand, this downward reference was present since the publication of RFC 2068 in 1997 ([RFC2068]), therefore it is unlikely to cause problems in practice. See also [BCP97].

[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.

[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.

## 10.2. Informative References

[BCP97]  Klensin, J. and S. Hartman, "Handling Normative References to Standards-Track Documents", BCP 97, RFC 4897, June 2007.

[RFC1945]  Berners-Lee, T., Fielding, R., and H. Nielsen, "Hypertext Transfer Protocol -- HTTP/1.0", RFC 1945, May 1996.

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

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

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

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

[RFC2295]  Holtman, K. and A. Mutz, "Transparent Content

Negotiation in HTTP", RFC 2295, March 1998.

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





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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

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

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

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

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC5322] Resnick, P., "Internet Message Format", RFC 5322, October 2008.

## Appendix A. Differences between HTTP and MIME

HTTP/1.1 uses many of the constructs defined for Internet Mail ([RFC5322]) and the Multipurpose Internet Mail Extensions (MIME [RFC2045]) to allow a message-body to be transmitted in an open variety of representations and with extensible mechanisms. However, RFC 2045 discusses mail, and HTTP has a few features that are different from those described in MIME. These differences were carefully chosen to optimize performance over binary connections, to allow greater freedom in the use of new media types, to make date

comparisons easier, and to acknowledge the practice of some early HTTP servers and clients.

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





## A.1. MIME-Version

HTTP is not a MIME-compliant protocol. However, HTTP/1.1 messages MAY include a single MIME-Version general-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 compliance with the MIME protocol (as defined in [RFC2045]). Proxies/gateways are responsible for ensuring full compliance (where possible) when exporting HTTP messages to strict MIME environments.

```
  MIME-Version   = "MIME-Version" ":" OWS MIME-Version-v
  MIME-Version-v = 1*DIGIT "." 1*DIGIT
```

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

## A.2. Conversion to Canonical Form

MIME requires that an Internet mail body-part be converted to canonical form prior to being transferred, as described in Section 4 of [RFC2049]. Section 2.3.1 of this document describes the forms allowed for subtypes of the "text" media type when transmitted over HTTP. [RFC2046] requires that content with a type of "text" represent line breaks as CRLF and forbids the use of CR or LF outside of line break sequences. HTTP allows CRLF, bare CR, and bare LF to indicate a line break within text content when a message is transmitted over HTTP.

Where it is possible, a proxy or gateway from HTTP to a strict MIME environment SHOULD translate all line breaks within the text media types described in Section 2.3.1 of this document to the RFC 2049 canonical form of CRLF. Note, however, that this might be complicated by the presence of a Content-Encoding and by the fact that HTTP allows the use of some character sets which do not use octets 13 and 10 to represent CR and LF, as is the case for some multi-byte character sets.

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 6.1 of [Part1]) to simplify the process of date comparison. Proxies and gateways from other protocols SHOULD ensure that any Date header





field present in a message conforms to one of the HTTP/1.1 formats and rewrite the date if necessary.

## A.4. Introduction of Content-Encoding

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

## A.5. No Content-Transfer-Encoding

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

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

### A.6. Introduction of Transfer-Encoding

HTTP/1.1 introduces the Transfer-Encoding header field (Section 9.7 of [Part1]). Proxies/gateways MUST remove any transfer-coding prior to forwarding a message via a MIME-compliant protocol.

### A.7. MHTML and Line Length Limitations

HTTP implementations which share code with MHTML [RFC2557] implementations need to be aware of MIME line length limitations. Since HTTP does not have this limitation, HTTP does not fold long lines. MHTML messages being transported by HTTP follow all conventions of MHTML, including line length limitations and folding, canonicalization, etc., since HTTP transports all message-bodies as payload (see Section 2.3.2) and does not interpret the content or any MIME header lines that might be contained therein.





## Appendix B. Additional Features

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

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

## Appendix C. Changes from RFC 2616

Clarify contexts that charset is used in. (Section 2.1)

Remove base URI setting semantics for Content-Location due to poor implementation support, which was caused by too many broken servers emitting bogus Content-Location header fields, and also the potentially undesirable effect of potentially breaking relative links in content-negotiated resources. (Section 6.7)

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

## Appendix D. Collected ABNF

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

Content-Encoding = "Content-Encoding:" OWS Content-Encoding-v
Content-Encoding-v = *( "," OWS ) content-coding *( OWS "," [ OWS
 content-coding ] )
Content-Language = "Content-Language:" OWS Content-Language-v
```





```
Content-Language-v = *( "," OWS ) language-tag *( OWS "," [ OWS
 language-tag ] )
Content-Length = <Content-Length, defined in [Part1], Section 9.2>
Content-Location = "Content-Location:" OWS Content-Location-v
Content-Location-v = absolute-URI / partial-URI
Content-MD5 = "Content-MD5:" OWS Content-MD5-v
Content-MD5-v = <base64 of 128 bit MD5 digest as per [RFC1864]>
Content-Range = <Content-Range, defined in [Part5], Section 5.2>
```

```
Content-Type = "Content-Type:" OWS Content-Type-v
Content-Type-v = media-type

Expires = <Expires, defined in [Part6], Section 3.3>

Last-Modified = <Last-Modified, defined in [Part4], Section 6.6>

MIME-Version = "MIME-Version:" OWS MIME-Version-v
MIME-Version-v = 1*DIGIT "." 1*DIGIT

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

absolute-URI = <absolute-URI, defined in [Part1], Section 2.6>
accept-ext = OWS ";" OWS token [ "=" word ]
accept-params = OWS ";" OWS "q=" qvalue *accept-ext
attribute = token

charset = token
codings = ( content-coding / "*" )
content-coding = token

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

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

parameter = attribute "=" value
partial-URI = <partial-URI, defined in [Part1], Section 2.6>

qvalue = <qvalue, defined in [Part1], Section 6.4>

subtype = token

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

value = word
```





```
word = <word, defined in [Part1], Section 1.2.2>
```

ABNF diagnostics:

```
; Accept defined but not used
; Accept-Charset defined but not used
; Accept-Encoding defined but not used
; Accept-Language defined but not used
; Content-Encoding defined but not used
; Content-Language defined but not used
; Content-Length defined but not used
; Content-Location defined but not used
; Content-MD5 defined but not used
; Content-Range defined but not used
; Content-Type defined but not used
; Expires defined but not used
; Last-Modified defined but not used
; MIME-Version defined but not used
```

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

### E.1. Since RFC 2616

Extracted relevant partitions from [RFC2616].

### E.2. Since draft-ietf-httpbis-p3-payload-00

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/8>: "Media Type Registrations" (<http://purl.org/NET/http-errata#media-reg>)
- <http://tools.ietf.org/wg/httpbis/trac/ticket/14>: "Clarification regarding quoting of charset values" (<http://purl.org/NET/http-errata#charactersets>)
- <http://tools.ietf.org/wg/httpbis/trac/ticket/16>: "Remove 'identity' token references" (<http://purl.org/NET/http-errata#identity>)
- <http://tools.ietf.org/wg/httpbis/trac/ticket/25>: "Accept-Encoding BNF"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/35>: "Normative and Informative references"
- <http://tools.ietf.org/wg/httpbis/trac/ticket/46>: "RFC1700 references"

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

### E.3. Since draft-ietf-httpbis-p3-payload-01

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

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

### E.4. Since draft-ietf-httpbis-p3-payload-02

Closed issues:

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

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

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

### E.5. Since draft-ietf-httpbis-p3-payload-03

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/67>: "Quoting Charsets"





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

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

Other changes:

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

### E.6. Since draft-ietf-httpbis-p3-payload-04

Closed issues:

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

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

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

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

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

### E.7. Since draft-ietf-httpbis-p3-payload-05

Closed issues:

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

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

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

Other changes:

- Move definition of quality values into Part 1.





## E.8. Since draft-ietf-httpbis-p3-payload-06

Closed issues:

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

- <http://tools.ietf.org/wg/httpbis/trac/ticket/155>: "Content Sniffing"

## E.9. Since draft-ietf-httpbis-p3-payload-07

Closed issues:

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

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

- <http://tools.ietf.org/wg/httpbis/trac/ticket/154>: "Content-Location base-setting problems"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/155>: "Content Sniffing"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/188>: "pick IANA policy (RFC5226) for Transfer Coding / Content Coding"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/189>: "move definitions of gzip/deflate/compress to part 1"

Partly resolved issues:

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

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

## E.10. Since draft-ietf-httpbis-p3-payload-08

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/81>: "Content Negotiation for media types"





- <http://tools.ietf.org/wg/httpbis/trac/ticket/181>: "Accept-Language: which RFC4647 filtering?"

## E.11. Since draft-ietf-httpbis-p3-payload-09

Closed issues:

- <http://tools.ietf.org/wg/httpbis/trac/ticket/122>: "MIME-Version not listed in P1, general header fields"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/143>: "IANA registry for content/transfer encodings"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/155>: "Content Sniffing"

- <http://tools.ietf.org/wg/httpbis/trac/ticket/200>: "use of term "word" when talking about header structure"

Partly resolved issues:

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

## E.12. Since draft-ietf-httpbis-p3-payload-10

Closed issues:

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





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

Partly resolved issues:

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

### E.13. Since draft-ietf-httpbis-p3-payload-11

Closed issues:

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

## Index

A

## Authors' Addresses

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
Alcatel-Lucent Bell Labs
21 Oak Knoll Road
Carlisle, MA  01741
USA

EMail: jg@freedesktop.org
URI:   http://gettys.wordpress.com/

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/