Network Working Group	R. Fielding, Ed.
Internet-Draft	Day Software
Obsoletes: 2616 (if approved)	J. Gettys
Intended status: Standards Track	One Laptop per Child
Expires: July 16, 2008	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
	January 13, 2008

TOC

HTTP/1.1, part 3: Message Payload and Content Negotiation
draft-ietf-httpbis-p3-payload-01

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.
Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.
Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as “work in progress.”
The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.
This Internet-Draft will expire on July 16, 2008.

Abstract

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information initiative since 1990. This document is Part 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://www.tools.ietf.org/wg/httpbis/trac/report/11 and related documents (including fancy diffs) can be found at http://www.tools.ietf.org/wg/httpbis/.
This draft incorporates those issue resolutions that were either collected in the original RFC2616 errata list (http://purl.org/NET/http-errata), or which were agreed upon on the mailing list between October 2006 and November 2007 (as published in "draft-lafon-rfc2616bis-03").

Table of Contents

1. Introduction
    1.1. Requirements
2. Protocol Parameters
    2.1. Character Sets
        2.1.1. Missing Charset
    2.2. Content Codings
    2.3. Media Types
        2.3.1. Canonicalization and Text Defaults
        2.3.2. Multipart Types
    2.4. Quality Values
    2.5. Language Tags
3. Entity
    3.1. Entity Header Fields
    3.2. Entity Body
        3.2.1. Type
        3.2.2. Entity Length
4. Content Negotiation
    4.1. Server-driven Negotiation
    4.2. Agent-driven Negotiation

4.3. Transparent Negotiation
5. Header Field Definitions
5.1. Accept
5.2. Accept-Charset
5.3. Accept-Encoding
5.4. Accept-Language
5.5. Content-Encoding
5.6. Content-Language
5.7. Content-Location
5.8. Content-MD5
5.9. Content-Type
6. IANA Considerations
7. Security Considerations
7.1. Privacy Issues Connected to Accept Headers
7.2. Content-Disposition Issues
8. Acknowledgments
9. References
9.1. Normative References
9.2. Informative References
Appendix A. Differences Between HTTP Entities and RFC 2045 Entities
A.1. MIME-Version
A.2. Conversion to Canonical Form
A.3. Introduction of Content-Encoding
A.4. No Content-Transfer-Encoding
A.5. Introduction of Transfer-Encoding
A.6. MHTML and Line Length Limitations
Appendix B. Additional Features
B.1. Content-Disposition
Appendix C. Compatibility with Previous Versions
C.1. Changes from RFC 2068
C.2. Changes from RFC 2616
Appendix D. Change Log (to be removed by RFC Editor before publication)
D.1. Since RFC2616
D.2. Since draft-ietf-httpbis-p3-payload-00
§ Index
§ Authors' Addresses
§ Intellectual Property and Copyright Statements

---

1. Introduction

TOC

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 may 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. The next 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] (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” June 1999.).

1.1. Requirements

TOC

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.).
An implementation is not compliant if it fails to satisfy one or more of the MUST or REQUIRED level requirements for the protocols it implements. An implementation that satisfies all the MUST or REQUIRED level and all the SHOULD level requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST level requirements but not all the SHOULD level requirements for its protocols is said to be "conditionally compliant."

---

2. Protocol Parameters

TOC

2.1. Character Sets

TOC

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 may be available in a given character set and a character set may 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 (in which the charset value is an unquoted token) and as the value of a parameter in a Content-Type header (within a request or response), in which case the parameter value of the charset parameter may be quoted.
Implementors should be aware of IETF character set requirements [RFC3629] (Yergeau, F., “UTF-8, a transformation format of ISO 10646,” November 2003.) [RFC2277] (Alvestrand, H., “IETF Policy on Character Sets and Languages,” January 1998.).

2.1.1. Missing Charset

TOC

Some HTTP/1.0 software has interpreted a Content-Type header 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] (International Organization for Standardization, “Information technology -- 8-bit single-byte coded graphic character sets -- Part 1: Latin alphabet No. 1,” 1998.)) 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 (Canonicalization and Text Defaults).

2.2. Content Codings

TOC

Content coding values indicate an encoding transformation that has been or can be applied to an entity. Content codings are primarily used to allow a document to be compressed or otherwise usefully transformed without losing the identity of its underlying media type and without loss of information. Frequently, the entity 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 5.3 (Accept-Encoding)) and Content-Encoding (Section 5.5 (Content-Encoding)) 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.
The Internet Assigned Numbers Authority (IANA) acts as a registry for content-coding value tokens. Initially, the registry contains the following tokens:
gzip

> An encoding format produced by the file compression program "gzip" (GNU zip) as described in [RFC1952] (Deutsch, P., Gailly, J-L., Adler, M., Deutsch, L., and G. Randers-Pehrson, “GZIP file format specification version 4.3,” May 1996.). This format is a Lempel-Ziv coding (LZ77) with a 32 bit CRC.

compress

> The encoding format produced by the common UNIX file compression program "compress". This format is an adaptive Lempel-Ziv-Welch coding (LZW).
>
> Use of program names for the identification of encoding formats is not desirable and is discouraged for future encodings. Their use here is representative of historical practice, not good design. For compatibility with previous implementations of HTTP, applications SHOULD consider "x-gzip" and "x-compress" to be equivalent to "gzip" and "compress" respectively.

deflate

> The "zlib" format defined in [RFC1950] (Deutsch, L. and J-L. Gailly, “ZLIB Compressed Data Format Specification version 3.3,” May 1996.) in combination with the "deflate" compression mechanism described in [RFC1951] (Deutsch, P., “DEFLATE Compressed Data Format Specification version 1.3,” May 1996.).

identity

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

New content-coding value tokens SHOULD be registered; to allow interoperability between clients and servers, specifications of the content coding algorithms needed to implement a new value SHOULD be publicly available and adequate for independent implementation, and conform to the purpose of content coding defined in this section.

2.3. Media Types

TOC

HTTP uses Internet Media Types [RFC2046] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types,” November 1996.) in the Content-Type (Section 5.9 (Content-Type)) and Accept (Section 5.1 (Accept)) header fields in order to provide open and extensible data typing and type negotiation.

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

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

```
parameter               = attribute "=" value
attribute               = token
value                   = token | quoted-string
```

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. Linear white space (LWS) MUST NOT be used between the type and subtype, nor between an attribute and its value. 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.
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] (Freed, N. and J. Klensin, “Media Type Specifications and Registration Procedures,” December 2005.). Use of non-registered media types is discouraged.

2.3.1. Canonicalization and Text Defaults

TOC

Internet media types are registered with a canonical form. An entity-body transferred via HTTP messages MUST be represented 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 entity-body. HTTP applications MUST accept CRLF, bare CR, and bare LF as being representative of a line break in text media received via HTTP. In addition, if the text is represented in a character set that does not use octets 13 and 10 for CR and LF respectively, as is the case for some multi-byte character sets, HTTP allows the use of whatever octet sequences are defined by that character set to represent the equivalent of CR and LF for line breaks. This flexibility regarding line breaks applies only to text media in the entity-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 an entity-body 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 set (Section 2.1 (Character Sets)) 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 sets other than "ISO-8859-1" or its subsets MUST be labeled with an appropriate charset value. See Section 2.1.1 (Missing Charset) for compatibility problems.

2.3.2. Multipart Types

TOC

MIME provides for a number of "multipart" types -- encapsulations of one or more entities within a single message-body. All multipart types share a common syntax, as defined in Section 5.1.1 of [RFC2046] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME)

Part Two: Media Types," November 1996.), 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. Unlike in RFC 2046, the epilogue of any multipart message MUST be empty; HTTP applications MUST NOT transmit the epilogue (even if the original multipart contains an epilogue). These restrictions exist in order to preserve the self-delimiting nature of a multipart message-body, wherein the "end" of the message-body is indicated by the ending multipart boundary.

In general, HTTP treats a multipart message-body no differently than any other media type: strictly as payload. The one exception is the "multipart/byteranges" type (Appendix A of [Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 5: Range Requests and Partial Responses," January 2008.)) when it appears in a 206 (Partial Content) response. In all other cases, 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.

In general, an HTTP user agent SHOULD follow the same or similar behavior as a MIME user agent would upon receipt of a multipart type. 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] (Masinter, L., "Returning Values from Forms: multipart/form-data," August 1998.).

2.4. Quality Values

TOC

HTTP content negotiation (Section 4 (Content Negotiation)) uses short "floating point" numbers to indicate the relative importance ("weight") of various negotiable parameters. A weight is normalized to a real number in the range 0 through 1, where 0 is the minimum and 1 the maximum value. If a parameter has a quality value of 0, then content with this parameter is `not acceptable' for the client. HTTP/1.1 applications MUST NOT generate more than three digits after the decimal point. User configuration of these values SHOULD also be limited in this fashion.

```
qvalue         = ( "0" [ "." 0*3DIGIT ] )
               | ( "1" [ "." 0*3("0") ] )
```

"Quality values" is a misnomer, since these values merely represent relative degradation in desired quality.

2.5. Language Tags

TOC

A language tag 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.
The syntax and registry of HTTP language tags is the same as that defined by [RFC1766] (Alvestrand, H., “Tags for the Identification of Languages,” March 1995.). In summary, a language tag is composed of 1 or more parts: A primary language tag and a possibly empty series of subtags:

```
language-tag  = primary-tag *( "-" subtag )
primary-tag   = 1*8ALPHA
subtag        = 1*8ALPHA
```

White space is not allowed within the tag and all tags are case-insensitive. The name space of language tags is administered by the IANA. Example tags include:

```
en, en-US, en-cockney, i-cherokee, x-pig-latin
```

where any two-letter primary-tag is an ISO-639 language abbreviation and any two-letter initial subtag is an ISO-3166 country code. (The last three tags above are not registered tags; all but the last are examples of tags which could be registered in future.)

---

3. Entity

TOC

Request and Response messages MAY transfer an entity if not otherwise restricted by the request method or response status code. An entity consists of entity-header fields and an entity-body, although some responses will only include the entity-headers.
In this section, both sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

TOC

3.1. Entity Header Fields

Entity-header fields define metainformation about the entity-body or, if no body is present, about the resource identified by the request.

```
       entity-header  = Allow                    ; [Part2], Section 10.1
                      | Content-Encoding         ; Section 5.5
                      | Content-Language         ; Section 5.6
                      | Content-Length           ; [Part1], Section 8.2
                      | Content-Location         ; Section 5.7
                      | Content-MD5              ; Section 5.8
                      | Content-Range            ; [Part5], Section 5.2
                      | Content-Type             ; Section 5.9
                      | Expires                  ; [Part6], Section 15.3
                      | Last-Modified            ; [Part4], Section 6.6
                      | extension-header

       extension-header = message-header
```

The extension-header mechanism allows additional entity-header fields to be defined without changing the protocol, but these fields cannot be assumed to be recognizable by the recipient. Unrecognized header fields SHOULD be ignored by the recipient and MUST be forwarded by transparent proxies.

3.2. Entity Body

TOC

The entity-body (if any) sent with an HTTP request or response is in a format and encoding defined by the entity-header fields.

```
       entity-body    = *OCTET
```

An entity-body is only present in a message when a message-body is present, as described in Section 4.3 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” January 2008.). The entity-body is obtained from the message-body by decoding any Transfer-Encoding that might have been applied to ensure safe and proper transfer of the message.

TOC

3.2.1. Type

When an entity-body is included with a message, the data type of that body is determined via the header fields Content-Type and Content-Encoding. These define a two-layer, ordered encoding model:

```
entity-body := Content-Encoding( Content-Type( data ) )
```

Content-Type specifies the media type of the underlying data. Content-Encoding may be used to indicate any additional content codings applied to the data, usually for the purpose of data compression, that are a property of the requested resource. There is no default encoding.
Any HTTP/1.1 message containing an entity-body SHOULD include a Content-Type header field defining the media type of that body. If and only if the media type is not given by a Content-Type field, the recipient MAY attempt to guess the media type via inspection of its content and/or the name extension(s) of the URI used to identify the resource. If the media type remains unknown, the recipient SHOULD treat it as type "application/octet-stream".

3.2.2. Entity Length

TOC

The entity-length of a message is the length of the message-body before any transfer-codings have been applied. Section 4.4 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” January 2008.) defines how the transfer-length of a message-body is determined.

---

4. Content Negotiation

TOC

Most HTTP responses include an entity which contains information for interpretation by a human user. Naturally, it is desirable to supply the user with the "best available" entity corresponding to the request. Unfortunately for servers and caches, not all users have the same preferences for what is "best," and not all user agents are equally capable of rendering all entity types. For that reason, HTTP has provisions for several mechanisms for "content negotiation" -- the process of selecting the best representation for a given response when there are multiple representations available.

> Note: This is not called "format negotiation" because the alternate representations may be of the same media type, but use different capabilities of that type, be in different languages, etc.

Any response containing an entity-body MAY be subject to negotiation, including error responses.
There are two kinds of content negotiation which are possible in HTTP: server-driven and agent-driven negotiation. These two kinds of negotiation are orthogonal and thus may be used separately or in combination. One method of combination, referred to as transparent negotiation, occurs when a cache uses the agent-driven negotiation information provided by the origin server in order to provide server-driven negotiation for subsequent requests.

4.1. Server-driven Negotiation

TOC

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 may 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 5.1 (Accept)), Accept-Charset (Section 5.2 (Accept-Charset)), Accept-Encoding (Section 5.3 (Accept-Encoding)), Accept-Language (Section 5.4 (Accept-Language)), and User-Agent (Section 10.9 of [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," January 2008.)). 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.
The Vary header field (Section 15.5 of [Part6] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., "HTTP/1.1, part 6: Caching," January 2008.)) can be used to express the parameters the server uses to select a representation that is subject to server-driven negotiation.

4.2. Agent-driven Negotiation

TOC

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 entity-body of the initial response, with each representation identified by its own URI. Selection from among the representations may 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.
HTTP/1.1 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.

4.3. Transparent Negotiation

TOC

Transparent negotiation is a combination of both server-driven and agent-driven negotiation. When a cache is supplied with a form of the list of available representations of the response (as in agent-driven negotiation) and the dimensions of variance are completely understood by the cache, then the cache becomes capable of performing server-driven negotiation on behalf of the origin server for subsequent requests on that resource.
Transparent negotiation has the advantage of distributing the negotiation work that would otherwise be required of the origin server and also removing the second request delay of agent-driven negotiation when the cache is able to correctly guess the right response.
This specification does not define any mechanism for transparent negotiation, though it also does not prevent any such mechanism from being developed as an extension that could be used within HTTP/1.1.

5. Header Field Definitions

TOC

This section defines the syntax and semantics of HTTP/1.1 header fields related to the payload of messages.
For entity-header fields, both sender and recipient refer to either the client or the server, depending on who sends and who receives the entity.

5.1. Accept

TOC

The Accept request-header field can be used to specify certain media types which are acceptable for the response. Accept headers 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" ":"
                 #( media-range [ accept-params ] )

media-range    = ( "*/*"
                 | ( type "/" "*" )
                 | ( type "/" subtype )
                 ) *( ";" parameter )
accept-params  = ";" "q" "=" qvalue *( accept-extension )
accept-extension = ";" token [ "=" ( token | quoted-string ) ]
```

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 2.4 (Quality Values)). 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 entity, and if that does not exist, send the text/plain entity." 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:

```
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.

5.2. Accept-Charset

TOC

The Accept-Charset request-header field can be used to indicate what character sets are acceptable for the response. 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" ":"
        1#( ( charset | "*" ) [ ";" "q" "=" qvalue ] )
```

Character set values are described in Section 2.1 (Character Sets). 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 is present, the default is that any character set is acceptable. If an Accept-Charset header is present, and if the server cannot send a response which is acceptable according to the Accept-Charset header, 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.

5.3. Accept-Encoding

TOC

The Accept-Encoding request-header field is similar to Accept, but restricts the content-codings (Section 2.2 (Content Codings)) that are acceptable in the response.

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

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 2.4 (Quality Values), 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, 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.

5.4. Accept-Language

TOC

The Accept-Language request-header field is similar to Accept, but restricts the set of natural languages that are preferred as a response to the request. Language tags are defined in Section 2.5 (Language Tags).

```
Accept-Language = "Accept-Language" ":"
                  1#( language-range [ ";" "q" "=" qvalue ] )
language-range  = ( ( 1*8ALPHA *( "-" 1*8ALPHA ) ) | "*" )
```

Each language-range MAY 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." A language-range matches a language-tag if it exactly equals the tag, or if it exactly equals a prefix of the tag such that the first tag character following the prefix is "-". The special range "*", if present in the Accept-Language field, matches every tag not matched by any other range present in the Accept-Language field.

> Note: This use of a prefix matching rule does not imply that language tags are assigned to languages in such a way that it is always true that if a user understands a language with a certain tag, then this user will also understand all languages with tags for which this tag is a prefix. The prefix rule simply allows the use of prefix tags if this is the case.

The language quality factor assigned to a language-tag by the Accept-Language field is the quality value of the longest language-range in the field that matches the language-tag. If no language-range in the field matches the tag, the language quality factor assigned is 0. If no Accept-Language header is present in the request, the server SHOULD assume that all languages are equally acceptable. If an Accept-Language header is present, then all languages which are assigned a quality factor greater than 0 are acceptable.
It might be contrary to the privacy expectations of the user to send an Accept-Language header with the complete linguistic preferences of the user in every request. For a discussion of this issue, see Section 7.1 (Privacy Issues Connected to Accept Headers).
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 should provide 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.

5.5. Content-Encoding

TOC

The Content-Encoding entity-header field is used as a modifier to the media-type. When present, its value indicates what additional content codings have been applied to the entity-body, 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 document to be compressed without losing the identity of its underlying media type.

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

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

```
Content-Encoding: gzip
```

The content-coding is a characteristic of the entity identified by the Request-URI. Typically, the entity-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 an entity is not "identity", then the response MUST include a Content-Encoding entity-header (Section 5.5 (Content-Encoding)) that lists the non-identity content-coding(s) used.
If the content-coding of an entity 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 an entity, 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 entity-header fields not defined by this specification.

5.6. Content-Language

TOC

The Content-Language entity-header field describes the natural language(s) of the intended audience for the enclosed entity. Note that

this might not be equivalent to all the languages used within the entity-body.

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

Language tags are defined in Section 2.5 (Language Tags). The primary purpose of Content-Language is to allow a user to identify and differentiate entities 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 an entity 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.

5.7. Content-Location

TOC

The Content-Location entity-header field MAY be used to supply the resource location for the entity enclosed in the message when that entity is accessible from a location separate from the requested resource's URI. A server SHOULD provide a Content-Location for the variant corresponding to the response entity; especially in the case where a resource has multiple entities associated with it, and those entities actually have separate locations by which they might be individually accessed, the server SHOULD provide a Content-Location for the particular variant which is returned.

```
Content-Location = "Content-Location" ":"
                   ( absoluteURI | relativeURI )
```

The value of Content-Location also defines the base URI for the entity. The Content-Location value is not a replacement for the original requested URI; it is only a statement of the location of the resource corresponding to this particular entity at the time of the request. Future requests MAY specify the Content-Location URI as the request-URI if the desire is to identify the source of that particular entity.
A cache cannot assume that an entity 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. However, the Content-Location can be used to differentiate between multiple entities retrieved from a single requested resource, as described in Section 7 of [Part6] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 6: Caching,” January 2008.).
If the Content-Location is a relative URI, the relative URI is interpreted relative to the Request-URI.
The meaning of the Content-Location header in PUT or POST requests is undefined; servers are free to ignore it in those cases.

5.8. Content-MD5

TOC

The Content-MD5 entity-header field, as defined in [RFC1864] (Myers, J. and M. Rose, “The Content-MD5 Header Field,” October 1995.), is an MD5 digest of the entity-body for the purpose of providing an end-to-end message integrity check (MIC) of the entity-body. (Note: a MIC is good for detecting accidental modification of the entity-body in transit, but is not proof against malicious attacks.)

```
Content-MD5   = "Content-MD5" ":" md5-digest
md5-digest    = <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 entity-body. Only origin servers or clients 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 of the entity-body, including gateways and proxies, MAY check that the digest value in this header field matches that of the entity-body as received.
The MD5 digest is computed based on the content of the entity-body, including any content-coding that has been applied, but not including any transfer-encoding applied to the message-body. If the message is received with a transfer-encoding, that encoding MUST be removed prior to checking the Content-MD5 value against the received entity.

This has the result that the digest is computed on the octets of the entity-body exactly as, and in the order that, they would be sent if no transfer-encoding were being applied.
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 entity-body for composite types MAY contain many body-parts, each with its own MIME and HTTP headers (including Content-MD5, Content-Transfer-Encoding, and Content-Encoding headers). If a body-part has a Content-Transfer-Encoding or Content-Encoding header, 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.

5.9. Content-Type

TOC

The Content-Type entity-header field indicates the media type of the entity-body sent to the recipient or, in the case of the HEAD method, the media type that would have been sent had the request been a GET.

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

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

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

Further discussion of methods for identifying the media type of an entity is provided in Section 3.2.1 (Type).

6. IANA Considerations

TOC

TBD.

7. Security Considerations

TOC

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

7.1. Privacy Issues Connected to Accept Headers

TOC

Accept request-headers can reveal information about the user to all servers which are accessed. The Accept-Language header 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 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 headers by default, and to ask the user whether or not to start sending Accept-Language headers 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 headers 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.

7.2. Content-Disposition Issues

TOC

[RFC1806] (Troost, R. and S. Dorner, “Communicating Presentation Information in Internet Messages: The Content-Disposition Header,” June 1995.), from which the often implemented Content-Disposition (see Appendix B.1 (Content-Disposition)) header in HTTP is derived, has a number of very serious security considerations. Content-Disposition is not part of the HTTP standard, but since it is widely implemented, we are documenting its use and risks for implementors. See [RFC2183] (Troost, R., Dorner, S., and K. Moore, “Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field,” August 1997.) (which updates [RFC1806] (Troost, R. and S. Dorner, “Communicating Presentation Information in Internet Messages: The Content-Disposition Header,” June 1995.)) for details.

---

8. Acknowledgments

TOC

---

9. References

TOC

9.1. Normative References

TOC

[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-01 (work in progress), January 2008.
[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-01 (work in progress), January 2008.
[Part4]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 4: Conditional Requests,” draft-ietf-httpbis-p4-conditional-01 (work in progress), January 2008.
[Part5]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 5: Range Requests and Partial Responses,” draft-ietf-httpbis-p5-range-01 (work in progress), January 2008.
[Part6]	Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 6: Caching,” draft-ietf-httpbis-p6-cache-01 (work in progress), January 2008.
[RFC1766]	Alvestrand, H., “Tags for the Identification of Languages,” RFC 1766, March 1995.
[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. RFC1950 is an Informational RFC, thus it may be less stable than this specification. On the other hand, this downward reference was present since [RFC2068] (Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” January 1997.) (published in 1997), therefore it is unlikely to cause problems in practice.
[RFC1951]	Deutsch, P., “DEFLATE Compressed Data Format Specification version 1.3,” RFC 1951, May 1996.

	RFC1951 is an Informational RFC, thus it may be less stable than this specification. On the other hand, this downward reference was present since [RFC2068] (Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," January 1997.) (published in 1997), therefore it is unlikely to cause problems in practice.
[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. RFC1952 is an Informational RFC, thus it may be less stable than this specification. On the other hand, this downward reference was present since [RFC2068] (Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," January 1997.) (published in 1997), therefore it is unlikely to cause problems in practice.
[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.
[RFC4288]	Freed, N. and J. Klensin, "Media Type Specifications and Registration Procedures," BCP 13, RFC 4288, December 2005.

9.2. Informative References

TOC

[RFC1806]	Troost, R. and S. Dorner, "Communicating Presentation Information in Internet Messages: The Content-Disposition Header," RFC 1806, June 1995.
[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.
[RFC2183]	

	Troost, R., Dorner, S., and K. Moore, “Communicating Presentation Information in Internet Messages: The Content-Disposition Header Field,” RFC 2183, August 1997.
[RFC2277]	Alvestrand, H., “IETF Policy on Character Sets and Languages,” BCP 18, RFC 2277, January 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.
[RFC2822]	Resnick, P., “Internet Message Format,” RFC 2822, April 2001.
[RFC3629]	Yergeau, F., “UTF-8, a transformation format of ISO 10646,” RFC 3629, STD 63, November 2003.

Appendix A. Differences Between HTTP Entities and RFC 2045 Entities

TOC

HTTP/1.1 uses many of the constructs defined for Internet Mail ([RFC2822] (Resnick, P., “Internet Message Format,” April 2001.)) and the Multipurpose Internet Mail Extensions (MIME [RFC2045] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,” November 1996.)) to allow entities 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 RFC 2045. 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 RFC 2045. 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.

TOC

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] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,” November 1996.)). Proxies/ gateways are responsible for ensuring full compliance (where possible) when exporting HTTP messages to strict MIME environments.

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

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

A.2. Conversion to Canonical Form

TOC

[RFC2045] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies,” November 1996.) requires that an Internet mail entity be converted to canonical form prior to being transferred, as described in Section 4 of [RFC2049] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples,” November 1996.). Section 2.3.1 (Canonicalization and Text Defaults) of this document describes the forms allowed for subtypes of the "text" media type when transmitted over HTTP. [RFC2046] (Freed, N. and N. Borenstein, “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types,” November 1996.) 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 (Canonicalization and Text Defaults) 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.

Implementors should note that 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. Introduction of Content-Encoding

TOC

RFC 2045 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 entity-body 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 RFC 2045).

A.4. No Content-Transfer-Encoding

TOC

HTTP does not use the Content-Transfer-Encoding field of RFC 2045. 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.5. Introduction of Transfer-Encoding

TOC

HTTP/1.1 introduces the Transfer-Encoding header field (Section 8.7 of [Part1] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 1: URIs, Connections, and Message Parsing,” January 2008.)). Proxies/gateways MUST remove any transfer-coding prior to forwarding a message via a MIME-compliant protocol.

A.6. MHTML and Line Length Limitations

TOC

HTTP implementations which share code with MHTML [RFC2557] (Palme, F., Hopmann, A., Shelness, N., and E. Stefferud, “MIME Encapsulation of

Aggregate Documents, such as HTML (MHTML),” March 1999.)
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 (Multipart Types)) and does not interpret the content or any MIME header lines that might be contained therein.

Appendix B. Additional Features

TOC

[RFC1945] (Berners-Lee, T., Fielding, R., and H. Nielsen, “Hypertext Transfer Protocol -- HTTP/1.0,” May 1996.) and [RFC2068] (Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” January 1997.) 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 headers, such as Content-Disposition and Title, from SMTP and MIME are also often implemented (see [RFC2076] (Palme, J., “Common Internet Message Headers,” February 1997.)).

B.1. Content-Disposition

TOC

The Content-Disposition response-header field has been proposed as a means for the origin server to suggest a default filename if the user requests that the content is saved to a file. This usage is derived from the definition of Content-Disposition in [RFC1806] (Troost, R. and S. Dorner, “Communicating Presentation Information in Internet Messages: The Content-Disposition Header,” June 1995.).

```
content-disposition = "Content-Disposition" ":"
                      disposition-type *( ";" disposition-parm )
disposition-type = "attachment" | disp-extension-token
disposition-parm = filename-parm | disp-extension-parm
filename-parm = "filename" "=" quoted-string
disp-extension-token = token
disp-extension-parm = token "=" ( token | quoted-string )
```

An example is

Content-Disposition: attachment; filename="fname.ext"

The receiving user agent SHOULD NOT respect any directory path information present in the filename-parm parameter, which is the only parameter believed to apply to HTTP implementations at this time. The filename SHOULD be treated as a terminal component only.
If this header is used in a response with the application/octet-stream content-type, the implied suggestion is that the user agent should not display the response, but directly enter a `save response as...' dialog.
See Section 7.2 (Content-Disposition Issues) for Content-Disposition security issues.

Appendix C. Compatibility with Previous Versions

TOC

C.1. Changes from RFC 2068

TOC

Transfer-coding and message lengths all interact in ways that required fixing exactly when chunked encoding is used (to allow for transfer encoding that may not be self delimiting); it was important to straighten out exactly how message lengths are computed. (Section 3.2.2 (Entity Length), see also [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,” January 2008.), [Part5] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 5: Range Requests and Partial Responses,” January 2008.) and [Part6] (Fielding, R., Ed., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T., Lafon, Y., Ed., and J. Reschke, Ed., “HTTP/1.1, part 6: Caching,” January 2008.)).
Charset wildcarding is introduced to avoid explosion of character set names in accept headers. (Section 5.2 (Accept-Charset))
Content-Base was deleted from the specification: it was not implemented widely, and there is no simple, safe way to introduce it without a robust extension mechanism. In addition, it is used in a similar, but not identical fashion in MHTML [RFC2557] (Palme, F., Hopmann, A., Shelness, N., and E. Stefferud, “MIME Encapsulation of Aggregate Documents, such as HTML (MHTML),” March 1999.).
A content-coding of "identity" was introduced, to solve problems discovered in caching. (Section 2.2 (Content Codings))

Quality Values of zero should indicate that "I don't want something" to allow clients to refuse a representation. (Section 2.4 (Quality Values))
The Alternates, Content-Version, Derived-From, Link, URI, Public and Content-Base header fields were defined in previous versions of this specification, but not commonly implemented. See [RFC2068] (Fielding, R., Gettys, J., Mogul, J., Nielsen, H., and T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1,” January 1997.).

C.2. Changes from RFC 2616 TOC

Clarify contexts that charset is used in. (Section 2.1 (Character Sets))
Remove reference to non-existant identity transfer-coding value tokens. (Appendix A.4 (No Content-Transfer-Encoding))

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

D.1. Since RFC2616 TOC

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

D.2. Since draft-ietf-httpbis-p3-payload-00 TOC

Closed issues:

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/8: "Media Type Registrations" (http://purl.org/NET/http-errata#media-reg)

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/14: "Clarification regarding quoting of charset values" (http://purl.org/NET/http-errata#charactersets)

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/16: "Remove 'identity' token references" (http://purl.org/NET/http-errata#identity)

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/25: "Accept-Encoding BNF"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/35: "Normative and Informative references"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/46: "RFC1700 references"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/65: "Informative references"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/66: "ISO-8859-1 Reference"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/68: "Encoding References Normative"

* http://www3.tools.ietf.org/wg/httpbis/trac/ticket/86: "Normative up-to-date references"

Index

TOC

A	
	Accept header
	Accept-Charset header
	Accept-Encoding header
	Accept-Language header
	Alternates header
C	
	compress
	Content-Base header
	Content-Disposition header
	Content-Encoding header
	Content-Language header
	Content-Location header
	Content-MD5 header
	Content-Type header
	Content-Version header
D	
	deflate
	Derived-From header

G	
	Grammar
	Accept
	Accept-Charset
	Accept-Encoding
	accept-extension
	Accept-Language
	accept-params
	attribute
	charset
	codings
	content-coding
	content-disposition
	Content-Encoding
	Content-Language
	Content-Location
	Content-MD5
	Content-Type
	disp-extension-parm
	disp-extension-token
	disposition-parm
	disposition-type
	entity-body
	entity-header
	extension-header
	filename-parm
	language-range
	language-tag
	md5-digest
	media-range
	media-type
	MIME-Version
	parameter
	primary-tag
	qvalue
	subtag
	subtype
	type
	value
	gzip
H	
	Headers
	Accept
	Accept-Charset
	Accept-Encoding
	Accept-Language

	Alternate
	Content-Base
	Content-Disposition
	Content-Encoding
	Content-Language
	Content-Location
	Content-MD5
	Content-Type
	Content-Version
	Derived-From
	Link
	Public
	URI
I	
	identity
L	
	Link header
P	
	Public header
U	
	URI header

Authors' Addresses

TOC

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

	1501 Page Mill Road, MS 1177
	Palo Alto, CA 94304
	USA
Email:	JeffMogul@acm.org
	Henrik Frystyk Nielsen
	Microsoft Corporation
	1 Microsoft Way
	Redmond, WA 98052
	USA
Email:	henrikn@microsoft.com
	Larry Masinter
	Adobe Systems, Incorporated
	345 Park Ave
	San Jose, CA 95110
	USA
Email:	LMM@acm.org
URI:	http://larry.masinter.net/
	Paul J. Leach
	Microsoft Corporation
	1 Microsoft Way
	Redmond, WA 98052
Email:	paulle@microsoft.com
	Tim Berners-Lee
	World Wide Web Consortium
	MIT Computer Science and Artificial Intelligence Laboratory
	The Stata Center, Building 32
	32 Vassar Street
	Cambridge, MA 02139
	USA
Email:	timbl@w3.org
URI:	http://www.w3.org/People/Berners-Lee/
	Yves Lafon (editor)
	World Wide Web Consortium
	W3C / ERCIM
	2004, rte des Lucioles
	Sophia-Antipolis, AM 06902
	France
Email:	ylafon@w3.org
URI:	http://www.raubacapeu.net/people/yves/
	Julian F. Reschke (editor)
	greenbytes GmbH

	Hafenweg 16
	Muenster, NW 48155
	Germany
Phone:	+49 251 2807760
Fax:	+49 251 2807761
Email:	julian.reschke@greenbytes.de
URI:	http://greenbytes.de/tech/webdav/

Full Copyright Statement

TOC

Copyright © The IETF Trust (2008).
This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.
This document and the information contained herein are provided on an “AS IS” basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY, THE IETF TRUST AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Intellectual Property

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.
Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.
The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.