          HTTP/1.1, part 3: Message Payload and Content Negotiation
                    draft-ietf-httpbis-p3-payload-08

Status of this Memo

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 April 29, 2010.

Copyright Notice

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/11> 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.9.

Internet-Draft   HTTP/1.1, Part 3   October 2009

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 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].

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 of entities in any
   response can be negotiated (including error responses).

entity

   The information transferred as the payload of a request or
   response. An entity consists of metadata in the form of entity-
   header fields and content in the form of an entity-body.

representation

   An entity included with a response that is subject to content
   negotiation. There may exist multiple representations associated
   with a particular response status.

variant

   A resource may have one, or more than one, representation(s)
   associated with it at any given instant. Each of these
   representations is termed a `variant'.  Use of the term `variant'
   does not necessarily imply that the resource is subject to content

   negotiation.

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]:

     quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
     token   = <token, 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>
     header-field   = <header-field, defined in [Part1], Section 3.2>
     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 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] [RFC2277].

2.1.1. Missing Charset

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]) 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 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) and Content-Encoding (Section 5.5) 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, and SHOULD NOT be used in the Content-Encoding
      header.

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:

   o Name

   o Description

   o Pointer to specification text

   Values to be added to this name space require expert review and a
   specification (see "Expert Review" and "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 5.9) and Accept (Section 5.1) 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           = 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. 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 may 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. 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) 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 for compatibility problems.

2.3.2. Multipart Types

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], 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]) 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].

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. Entity

Request and Response messages MAY transfer an entity if not otherwise
restricted by the request method or response status code. An entity
consists of entity-header fields and an entity-body, although some
responses will only include the entity-headers.

In this section, both sender and recipient refer to either the client
or the server, depending on who sends and who receives the entity.

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  = Content-Encoding          ; Section 5.5
                   / Content-Language          ; Section 5.6
                   / Content-Length            ; [Part1], Section 9.2
                   / Content-Location          ; Section 5.7
                   / Content-MD5               ; Section 5.8
                   / Content-Range             ; [Part5], Section 5.2
                   / Content-Type              ; Section 5.9
                   / Expires                   ; [Part6], Section 3.3
                   / Last-Modified             ; [Part4], Section 6.6
                   / extension-header

    extension-header = header-field
```

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

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 3.3 of [Part1]. 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.

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. Any
HTTP/1.1 message containing an entity-body SHOULD include a Content-
Type header field defining the media type of that body, unless that
information is unknown. If the Content-Type header field is not
present, it indicates that the sender does not know the media type of
the data; recipients MAY either assume that it is "application/
octet-stream" ([RFC2046], Section 4.5.1) or examine the content to
determine its type.

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.

3.2.2. Entity Length

The entity-length of a message is the length of the message-body
before any transfer-codings have been applied. Section 3.4 of
[Part1] defines how the transfer-length of a message-body is
determined.

Fielding, et al.   Expires April 29, 2010   [Page 13]

4. Content Negotiation

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

    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-
   Charset ([Section 5.2](#)), Accept-Encoding ([Section 5.3](#)), Accept-Language
   ([Section 5.4](#)), 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.

   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.

[4.2](#). 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 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

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

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

   The "Accept" request-header field can be used by user agents to
   specify response media types that are acceptable. 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" ":" 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
                     [ "=" ( 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 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 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:

```
+-------------------+---------------+
| 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.

5.2. 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 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

Fielding, et al.   Expires April 29, 2010   [Page 19]

Internet-Draft   HTTP/1.1, Part 3   October 2009

   the sending of an unacceptable response is also allowed.

5.3. 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, 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

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."

For matching, the "Basic Filtering" matching scheme, defined in

Fielding, et al.   Expires April 29, 2010   [Page 21]

Internet-Draft   HTTP/1.1, Part 3   October 2009

Section 3.3.1 of [RFC4647], is used:

A language range matches a particular language tag if, in a case-
insensitive comparison, it exactly equals the tag, or if it
exactly equals a prefix of the tag such that the first 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](#).

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

   The "Content-Encoding" entity-header field indicates what 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" ":" 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 entity identified by
   the request-target. 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)
   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

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

       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

   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

   The "Content-Location" entity-header field is used to supply a URI
   for the entity in the message when it 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" ":" OWS

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

   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-
   target if the desire is to identify the source of that particular
   entity.

   Section 6.1 of [Part2] describes how clients may process the Content-
   Location header field.

   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
   2.6 of [Part6].

   If the Content-Location is a relative URI, the relative URI is
   interpreted relative to the request-target.

   The meaning of the Content-Location header in requests is undefined;
   servers are free to ignore it in those cases.

5.8. Content-MD5

   The "Content-MD5" entity-header field, as defined in [RFC1864], is an
   MD5 digest of the entity-body that provides an end-to-end message
   integrity check (MIC) of the entity-body. Note that 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" ":" 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 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

The "Content-Type" entity-header field indicates the media type of
the entity-body. 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 methods for identifying the media type of an
entity is provided in Section 3.2.1.

6. IANA Considerations

6.1. Message Header Registration

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

Header Field Name	Protocol	Status	Reference
Accept	http	standard	Section 5.1
Accept-Charset	http	standard	Section 5.2
Accept-Encoding	http	standard	Section 5.3
Accept-Language	http	standard	Section 5.4
Content-Disposition	http		Appendix B.1
Content-Encoding	http	standard	Section 5.5

```
   | Content-Language     | http      | standard | Section 5.6   |
   | Content-Location     | http      | standard | Section 5.7   |
   | Content-MD5          | http      | standard | Section 5.8   |
   | Content-Type         | http      | standard | Section 5.9   |
   | MIME-Version         | http      |          | Appendix A.1  |
   +---------------------+---------+---------+-------------+
```

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

6.2. 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> should be updated
   with the registration below:

Name	Description	Reference
compress	UNIX "compress" program method	Section 6.2.2.1 of
		[Part1]
deflate	"zlib" format [RFC1950] with	Section 6.2.2.2 of
	"deflate" compression	[Part1]
gzip	Same as GNU zip [RFC1952]	Section 6.2.2.3 of
		[Part1]
identity	No transformation	Section 2.2

7. 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.

7.1. Privacy Issues Connected to Accept Headers

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

   [RFC2183], from which the often implemented Content-Disposition (see
   Appendix B.1) 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 Section 5 of [RFC2183] for
   details.

8. Acknowledgments

9. References

9.1. 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-08
             (work in progress), October 2009.

   [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-08 (work in
             progress), October 2009.

   [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-08 (work in
             progress), October 2009.

[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-08 (work
            in progress), October 2009.

[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-08 (work in
            progress), October 2009.

[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 may 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 may 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.

9.2. 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.

   [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.

   [RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO
              10646", RFC 3629, STD 63, 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 Entities and RFC 2045 Entities

   HTTP/1.1 uses many of the constructs defined for Internet Mail
   ([RFC5322]) and the Multipurpose Internet Mail Extensions (MIME
   [RFC2045]) 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.

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

[RFC2045] requires that an Internet mail entity 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.

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.

. 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

   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.5. No Content-Transfer-Encoding

   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.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 headers, such as Content-Disposition and Title,
   from SMTP and MIME are also often implemented (see [RFC2076]).

B.1. Content-Disposition

   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 [RFC2183].

      content-disposition = "Content-Disposition" ":" OWS
                            content-disposition-v
      content-disposition-v = disposition-type
                            *(OWS ";" OWS 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](#) for Content-Disposition security issues.

Appendix C. Compatibility with Previous Versions

C.1. Changes from [RFC 2068](#)

   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](#), see also [[Part1](#)], [[Part5](#)] and [[Part6](#)]).

   Charset wildcarding is introduced to avoid explosion of character set
   names in accept headers. ([Section 5.2](#))

   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](#)].

   A content-coding of "identity" was introduced, to solve problems
   discovered in caching. ([Section 2.2](#))

   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 Section 19.6.2 of [RFC2068].

C.2. 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 headers, and also the potentially undesirable effect of potentially breaking relative links in content-negotiated resources. (Section 5.7)

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 [ "=" ( token / quoted-string ) ]
accept-params = OWS ";" OWS "q=" qvalue *accept-ext
attribute = token

charset = token
codings = ( content-coding / "*" )
content-coding = token
content-disposition = "Content-Disposition:" OWS
 content-disposition-v
content-disposition-v = disposition-type *( OWS ";" OWS
 disposition-parm )

disp-extension-parm = token "=" ( token / quoted-string )
disp-extension-token = token
disposition-parm = filename-parm / disp-extension-parm
disposition-type = "attachment" / disp-extension-token

entity-body = *OCTET
entity-header = Content-Encoding / Content-Language / Content-Length
 / Content-Location / Content-MD5 / Content-Range / Content-Type /
 Expires / Last-Modified / extension-header
extension-header = header-field

filename-parm = "filename=" quoted-string

header-field = <header-field, defined in [Part1], Section 3.2>
```

```
    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>

    quoted-string = <quoted-string, defined in [Part1], Section 1.2.2>
    qvalue = <qvalue, defined in [Part1], Section 6.4>

    subtype = token

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

    value = token / quoted-string

    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
    ; MIME-Version defined but not used
    ; content-disposition defined but not used
    ; entity-body defined but not used
    ; entity-header defined but not used
```

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

E.1. Since RFC2616

    Extracted relevant partitions from [RFC2616].

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

    Closed issues:

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

    o <http://tools.ietf.org/wg/httpbis/trac/ticket/14>: "Clarification

regarding quoting of charset values"
(<http://purl.org/NET/http-errata#charactersets>)

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

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/25>: "Accept-
      Encoding BNF"

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/35>: "Normative and
      Informative references"

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/46>: "RFC1700
      references"

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/55>: "Updating to
      RFC4288"

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/65>: "Informative
      references"

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/66>: "ISO-8859-1
      Reference"

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/68>: "Encoding
      References Normative"

   o <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>):

   o 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:

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

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/105>:
      "Classification for Allow header"

   o <http://tools.ietf.org/wg/httpbis/trac/ticket/115>: "missing
      default for qvalue in description of Accept-Encoding"

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

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

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

   Closed issues:

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

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

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

   Other changes:

   o <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:

   o <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>):

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

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

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

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

   Closed issues:

   o <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>):

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

   Other changes:

   o Move definition of quality values into Part 1.

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

   Closed issues:

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

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

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

Closed issues:

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

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

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

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

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

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

Partly resolved issues:

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

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

Index

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
    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/